EXTREME PROGRAMMING OHJELMISTOKEHITYKSESSÄ

Case: Tietokonepakettilaskurisovellus (Data Group Jyväskylä)

Eeva Hänninen

Opinnäytetyö
Huhtikuu 2010

Tietojenkäsittely
Luonnontieteiden ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖN
KUVAILULEHTI

Tekijä(t)	Julkaisun laji	Päivämäärä
HÄNNINEN, Eeva	Opinnäytetyö	20.04.2010
	Sivumäärä 61	Julkaisun kieli Suomi
	Luottamuksellisuus ( ) saakka	Verkkojulkaisulupa myönnetty (X)

Työn nimi
EXTREME PROGRAMMING OHJELMISTOKEHITYKSESSÄ
Case: Tietokonepakettilaskurisovellus (Data Group Jyväskylä)

Koulutusohjelma
Tietojenkäsittely

Työn ohjaaja(t)
KIVIAHO, Niko

Toimeksiantaja(t)
Data Group Jyväskylä

Tiivistelmä

Opinnäytetyön toimeksiantajana oli Data Group Jyväskylä. Opinnäytetyön tarkoituksena oli tutustua ketteriin menetelmiin ohjelmistotuotannossa sekä toteuttaa web-sovellus yhtä menetelmää käyttäen. Sovelluksen tarkoituksena oli olla tietokonepakettilaskuri, jonka kautta käyttäjä pystyy tilaamaan tietokoneen tiettyine komponentteineen, oheislaitteineen, ohjelmistoineen ja lisäpalveluineen. Työn ominaisuuksien pohjalta tutkimusmenetelmäksi valittiin kehityshanke.

Työn teoriaosuudessa tutustuttiin alan kirjallisuuteen ketteristä menetelmistä sekä esiteltiin niistä kolme: Scrum, Crystal ja Extreme Programming (XP). XP esiteltiin tarkemmin omassa luvussaan, jossa tutustuttiin myös kahteen XP:llä toteutettuun projektiin ja niistä saatuihin tuloksiin. Sovellus toteutettiin käytetyn ketterän menetelmän, XP:n, prosessin mukaisessa järjestyksessä. Sovelluksen toteutus ja sen kautta saadut kokemukset dokumentoitiin työhön.

XP:n soveltuvuutta erilaisiin projekteihin pohdittiin työn teoriaosuuden ja kehityshankkeen kautta saatujen kokemuksien pohjalta. Pohdinnan kautta ilmeni, että XP:n eri käytäntöjä voidaan hyödyntää laajasti monenlaisissa projekteissa. Kaikkien käytäntöjen hyödyntäminen ei ole este XP:n käytölle. XP:n käytäntöjä voidaan soveltaa myös projekteihin, joissa tekijöitä on vain yksi. Tällaisessa tapauksessa tekijältä vaaditaan kuitenkin hyvää teknistä taitoa, jotta eri menetelmiä voidaan hyödyntää. Parhaimmillaan XP on projektissa, jossa XP on tuttu sekä projektiryhmälle että asiakkaalle.

Avainsanat (asiasanat)
ketterät menetelmät, Extreme Programming, ohjelmistotuotanto

Muut tiedot

Author(s)
HÄNNINEN, Eeva

Type of publication
Bachelor´s Thesis

Date
20042010

Pages
61

Language
Finnish

Confidential

( ) Until

Permission for web publication
(X)

Title
EXTREME PROGRAMMING IN SOFTWARE DEVELOPMENT
Case: Software for Computer Packet Calculator (Data Group Jyväskylä)

Degree Programme
Business Information Systems

Tutor(s)
KIVIAHO, Niko

Assigned by
Data Group Jyväskylä

Abstract

The thesis was assigned by Data Group Jyväskylä. The purpose of this thesis was to study agile methods in software development and carry out a project using one of the methods. The aim of the project was to develop a web application. The purpose of the web application was to be a calculator with which a user can create and order a computer packet including certain components, peripherals, software and additional services. Due to these features the research method was selected to be a development project.

In the theory part of the work agile methods were presented generally. The focus was on three methods: Scrum, Crystal and Extreme Programming (XP). XP was presented widely in its own chapter where two XP projects were also examined. The web application was developed using XP. The implementation and experiences gathered during it were documented in this work.

The suitability of XP in different kind or projects was considered via theory of agile methods and the experiences of the web application project. It turned out that XP can be adapted fully or some parts of it can be used widely in many kind of projects. The practices of XP can be applied also to projects with only one person. In these cases good technical skills are required from the person in order to get the benefit from different practices. XP can be adopted best when it is familiar to the project group and to the customer.

Keywords
agile methods, Extreme Programming, software development

Miscellaneous

SISÄLTÖ

KUVIOT

TAULUKOT

ASIASANASTO

.NET Framework	Microsoftin kehittämä ohjelmointikehys, joka sisältää useita valmiita komponentteja, ratkaisuja ja kirjastoja.
ASP.NET	.NET-ohjelmointikehyksen laajennus dynaamisten web-sivujen, -sovellusten ja -palveluiden luomiseen.
Inkrementaalinen	Kumuloituva, lisääntyvä
Iteratiivinen	Vaiheittainen, jaksottainen
MVC-malli	Ohjelmistoarkkitehtuurityyli, jonka tarkoituksena on käyttöliittymän erottaminen sovellusaluetiedosta.
Refaktorointi	Prosessi, jossa tietokoneohjelman lähdekoodia muutetaan siten, että sen toiminnallisuus säilyy, mutta sen sisäinen rakenne paranee.
Sprintti	Kuvaa 1-4 viikon mittaista iteraatiota Scrum-menetelmässä.
WebForms	ASP.NETissä käytetty kontrolli- ja tapahtumapohjainen malli, joissa eri kontrolleihin voidaan sisällyttää HTML-kieltä, Java Scriptiä tai CSS-tyylittelyjä.
XP	Extreme Programming

1 OHJELMISTOKEHITYS KETTERIEN MENETELMIEN AVULLA

Perinteisiä ohjelmistokehityksen prosessimalleja on käytetty 1970-luvulta saakka. Yleisin näistä prosessimalleista on vesiputousmalli. Mallista on olemassa useita eri muunnelmia, mutta yleensä niistä voidaan erottaa ainakin määrittely-, suunnittelu- ja toteutusvaiheet. Vesiputousmallin mukaisessa prosessissa eri vaiheet seuraavat toisiaan, kunnes lopputuote on valmis ja käyttöönotto ja ylläpito -vaihe aloitetaan. Jokaisen vaiheen tuotos on seuraavan vaiheen aloitusperuste. (Haikala & Märijärvi 2004, 15, 36–37.) Vesiputousmallin etuna on sen ennustettavuus; prosessin alussa luodaan aikataulusuunnitelma, jonka mukaan projektia viedään eteenpäin. Aikataulusuunnitelma toimii pohjana kustannusten laskennalle. Vesiputousmalli on toimiva ratkaisu silloin, kun prosessin aikana ei tule yllätyksiä, vaan kaikki sujuu suunnitelmien mukaan. Tyypillisesti ohjelmistoprojektit eivät kuitenkaan ole toiminnoiltaan, aikataulultaan ja kustannuksiltaan helposti ennakoitavissa, vaan muutoksia ja ongelmia ilmenee kehitystyön aikana. Tällöin vesiputousmalli käy menetelmänä raskaaksi, sillä sen aikaisempiin vaiheisiin palaaminen on hidasta ja työlästä.

Tekniikan kehitys ja sitä myöten suurempien ja haastavampien ohjelmistoprojektien lisääntyminen on pakottanut ohjelmistokehittäjät miettimään prosessimalleja uudelleen. Yhdysvalloissa vuonna 2001 joukko uusien ohjelmistokehitysmenetelmien kehittäjiä kokoontui miettimään, mitä yhteistä heidän kehittämillään menetelmillä oli. Tämän tapaamisen aikana määriteltiin ketterät menetelmät. (Peltoniemi 2009, 43–44; Serena Software 2007, 3.) Ketterien menetelmien yhteinen idea on tuottaa asiakkaalle toimivaa ja laadukasta ohjelmistoa lyhyin väliajoin. Tätä kautta pyritään hallitsemaan prosessin aikana muuttuvia vaatimuksia sekä toimimaan tiiviissä yhteistyössä asiakkaan kanssa.

Tässä opinnäytetyössä tarkastellaan yleisesti ketteriä ohjelmistokehitysmenetelmiä sekä esitellään niistä kolme: Scrum- Extreme Programming (XP)- ja Crystal-menetelmä. XP, on kuvattu omassa luvussaan, jossa perehdytään sen arvoihin, elinkaareen, rooleihin, käytäntöihin ja prosesseihin. Samassa luvussa tarkastellaan myös kahta projektia, jotka toteutettiin XP:tä hyväksi käyttäen.

Opinnäytetyön toisessa osassa kuvataan XP-menetelmää hyödyntäen tehdyn sovelluksen toteutus. Sovelluksen toimeksiantajana toimii Data Group Jyväskylä, jonne opinnäytetyön tekijä suoritti työharjoittelun loppuvuonna 2009. Työharjoittelun aikana ilmeni, että toimeksiantaja haluaisi kehittää myymälätoimintaansa ja aktivoida asiakkaita. Keskustelujen pohjalta syntyi idea web-sovelluksesta, tietokonepakettilaskurista, jonka tekijä päätti toteuttaa opinnäytetyönä työharjoittelun jälkeen. Sovelluksen taustat ja tavoitteet on kuvattu luvussa 2.2.

Opinnäytetyön kahden ensimmäisen tutkimuskysymyksen kautta tutustutaan XP-menetelmiin sekä havainnollistetaan XP:n käyttöä ohjelmistoprojekteissa muutamien esimerkkitapauksien kautta. Näihin tutkimuskysymyksiin pyritään löytämään vastaukset opinnäytetyön teoriaosuudesta. Kolmas tutkimuskysymys käsittelee XP:n käyttöä tekijän toimeksiantajalle toteutettavan sovelluksen kehitystyössä. Tässä käytännön osuudessa tullaan yhdessä pohdinnan kanssa tekemään johtopäätöksiä siitä, miten XP soveltui tämänkaltaiseen projektiin, mitkä olivat sen tuomat edut ja toisaalta ongelmat. Opinnäytetyön tutkimuskysymykset on kokonaisuudessaan listattu luvussa 2.4.

2 TUTKIMUSASETELMA

2.1 Toimeksiantaja

Tämän opinnäytetyön toimeksiantajana toimii Data Group Jyväskylä. Data Group Jyväskylä on atk-alan erikoisliike, joka myy tuote- ja palveluratkaisuja kuluttajille sekä yrityksille. Toimeksiantajan toimipiste sijaitsee aivan Jyväskylän keskustassa. Toimipisteessä on atk-myymälän lisäksi huolto sekä yritysmyynnin puoli.

2.2 Työn taustat ja tavoitteet

Opinnäytetyön aihe löytyi toimeksiantajalle suoritetun työharjoittelun perusteella. Tekijä suoritti työharjoittelun loppuvuonna 2009 pääasiassa myymälän tehtävissä. Harjoittelun aikana keskusteltiin myös opinnäytetyön tekemisestä toimeksiantajalle. Toimeksiantaja ilmaisi halukkuutensa kehittää myymälän toimintaa. Myymälässä myytäviä tuotteita ovat muun muassa tietokoneet, kannettavat tietokoneet, näytöt, ohjelmistot, oheislaitteet ja muut tarvikkeet. Tekijän oman kiinnostuksen ja toimeksiantajan kanssa käytyjen keskustelujen pohjalta tultiin siihen tulokseen, että tietokonepakettilaskurin toteuttaminen web-sovelluksena olisi hyvä kehittämishanke. Koska opinnäytetyön tekeminen työharjoittelun ohessa ei ollut mahdollista, päätettiin sen tekeminen aloittaa työharjoittelun jälkeen, käytännössä alkuvuonna 2010.

Toimeksiantajalle toteutettava web-sovellus tulee olemaan tietokonepakettilaskuri, jonka kautta käyttäjä pystyy tilaamaan haluamansa laisen tietokoneen oheislaitteineen, ohjelmistoineen ja lisäpalveluineen. Toimeksiantaja voi halutessaan vaihtaa tietokonepaketin komponentteja, oheislaitteita, ohjelmistoja ja lisäpalveluita. Sovelluksen tarkoituksena on toimeksiantajaa lainaten ”herätellä asiakkaita”, tuoda lisäarvoa www-sivuille sekä edistää myymälän myyntiä. Tällä hetkellä toimeksiantaja ei tarjoa mahdollisuutta tilata räätälöityä tietokonepakettia suoraan www-sivuiltaan, vaan asiakkaiden tulee olla yhteydessä myymälään.

Ketterien menetelmien tutkimiseen päädyttiin tekijän omien kiinnostuksenkohteiden kautta. Esiteltävät ketterät menetelmät valittiin sen perusteella, kuinka suosittuja ne

ovat käytännössä. Ketteristä menetelmistä sovelluksen kehityksessä päädyttiin käyttämään XP-menetelmää, sillä se soveltuu ketteristä menetelmistä parhaiten tämän tyyliseen projektiin. Näin ollen opinnäytetyöhön tullaan havainnollistamaan XP:n käyttämistä käytännössä toteutettavan sovelluksen kehityksen kautta. Tässä opinnäytetyössä kuvataan sovelluksen teknisiä ominaisuuksia ja sisältöä kuvataan niiltä osin, kuin se on XP:n puolesta tarpeellista. Pääasiallinen tarkastelukulma asiaan on XP-menetelmän hyödyntäminen projektin aikana.

2.3 Tutkimusmenetelmä

Opinnäytetyö on tietokantapohjaisen web-sovelluksen kehityshanke. Tavoitteena on kartoittaa käytettävän ohjelmistokehityksen menetelmän, XP:n, vaatimukset ja kehittää web-sovellus niitä hyödyntäen. Työssä ilmenevät kehityshankkeelle tunnusomaiset piirteet: ongelmalähtöisyys, tavoitteellisuus, osallistuvuus, suunnitelmallisuus sekä kertaluonteisuus. Vaikka piirteisiin kuuluu kertaluonteisuus, voidaan web-sovellusta kopioida vastaavanlaiseen tarpeeseen, mutta tällöin sitä joudutaan räätälöimään tilanteeseen sopivaksi.

2.4 Tutkimuskysymykset

Tutkimuskysymyksiä ovat:

1. Millainen on Extreme Programming -menetelmä?
2. Minkälaisia kokemuksia Extreme Programming -menetelmä käytöstä on ohjelmistoprojekteissa?
3. Millä tavalla Extreme Programming -menetelmä soveltuu pienen web-sovelluksen toteuttamiseen?

3 KETTERÄT MENETELMÄT

Tässä luvussa esitellään ketterien menetelmien historiaa, tehdään vertailua perinteisiin menetelmiin sekä tutustutaan kahteen ketterään menetelmään. Ensimmäisessä alaluvussa tarkastellaan ketterien menetelmien historiaa ja yleisiä arvoja. Luku 3.2 sisältää perinteisten ja ketterien menetelmien vertailua. Ketteristä menetelmistä kaksi, Scrum ja Crystal, on esitelty luvuissa 3.3–3.4. Kolmas tunnettu ketterä menetelmä, Extreme Programming - malli, eli XP, esitellään laajemmin omassa luvussaan 3. Scrum ja XP valittiin sen takia, että ne ovat kaksi eniten käytettyä ketterää menetelmää. Crystal puolestaan eroaa hieman muista ketteristä menetelmistä, ja on tämän vuoksi tutustumisen arvoinen.

3.1 Ketterien menetelmien historia ja yleiset arvot

Tekniikan nopea kehitys on pakottanut ohjelmistokehittäjät arvioimaan toimintatapojaan uudelleen. Alkusysäys tälle tapahtui Yhdysvalloissa vuonna 2001, kun joukko uusien ohjelmistokehitysmenetelmien kehittäjiä kokoontui yhteen miettimään, mitä yhteistä heidän kehittämillään menetelmillä oli. Syntyi termi ”ketterä” (Agile), joka kuvastaa näiden menetelmien yhteisiä piirteitä. Kehittäjien mielestä ketterä ohjelmistokehitys ei ole ainoastaan yksi metodologia tai paketti työkaluja, vaan kokonainen filosofia. Samalla perustettiin myös voittoa tavoittelematon Agile Allience - järjestö sekä luotiin ketterän ohjelmistokehityksen manifesti. Tätä manifestia noudattavat kaikki yksittäiset menetelmät ja sen tärkeimmässä osassa määritellään yleiset arvot ketterälle ohjelmistokehitykselle. (Peltoniemi 2009, 43–44; Serena Software 2007, 3.) Vapaasti suomennettuna manifesti on seuraava:

Me etsimme parempia keinoja ohjelmistojen kehitykseen tekemällä niitä itse ja auttamalla siinä muita. Tämän kautta olemme päätyneet arvostamaan

- ***yksilöitä ja vuorovaikutusta*** *enemmän kuin prosesseja ja työkaluja*
- ***toimivaa ohjelmistoa*** *enemmän kuin kattavaa dokumentaatiota*
- ***asiakasyhteistyötä*** *enemmän kuin sopimusneuvotteluja*

- ***muutokseen reagoimista*** *enemmän kuin suunnitelman noudattamista.*

Vaikka oikealla puolella olevilla asioilla on arvoa, arvostamme vasemmalla puolella olevia asioita enemmän. (Beck, Beedle, Bennekum, Cockburn, Cunningham, Fowler, Grenning, Highsmith, Hunt, Jeffries, Kern, Marick, Martin, Mellor, Schwaber, Sutherland, & Thomas 2001.)

Abrahamsson, Salo, Ronkainen ja Warsta (2002) sekä Beck ja muut (2001) avaavat keskeisimpiä arvoja seuraavasti:

Yhteisöllisyyden korostaminen. Ketterä ohjelmistokehitys korostaa ohjelmistokehittäjien yhteisöllisyyttä sekä yksittäisen henkilön asemaa prosessissa. Käytännössä manifesti kehottaa luomaan hyvän työyhteisön ja ylläpitämään sitä, järjestämään tarvittavan työympäristön ja työkalut sekä kehittämään ja ylläpitämään niitä toimintatapoja, joilla ryhmähenkeä saadaan kasvatettua. Kehitystiimin tulee myös säännöllisin väliajoin pysähtyä tarkastelemaan omaa toimintaansa ja miettimään, kuinka sitä saataisiin tehostettua, ja tämän jälkeen soveltaa näitä keinoja käytäntöön. (Abrahamsson ym. 2002, 11–12; Beck ym. 2001.)

Toimivan ja testatun ohjelman tuottaminen. Sovelluskehitystiimin tärkein tavoite on jatkuvasti tuottaa toimivaa ja testattua sovellusta, jolla on arvoa asiakkaalle. Uudet julkaisut tuotetaan säännöllisin väliajoin, joissain tapauksissa esimerkiksi tunnin tai päivän, mutta yleensä kahden–neljän viikon välein. Kehittäjiä kannustetaan pitämään koodi yksinkertaisena ja teknisesti laadukkaana. Dokumentaatiota tuotetaan vain asianmukainen määrä. (Abrahamsson ym. 2002, 11–12.)

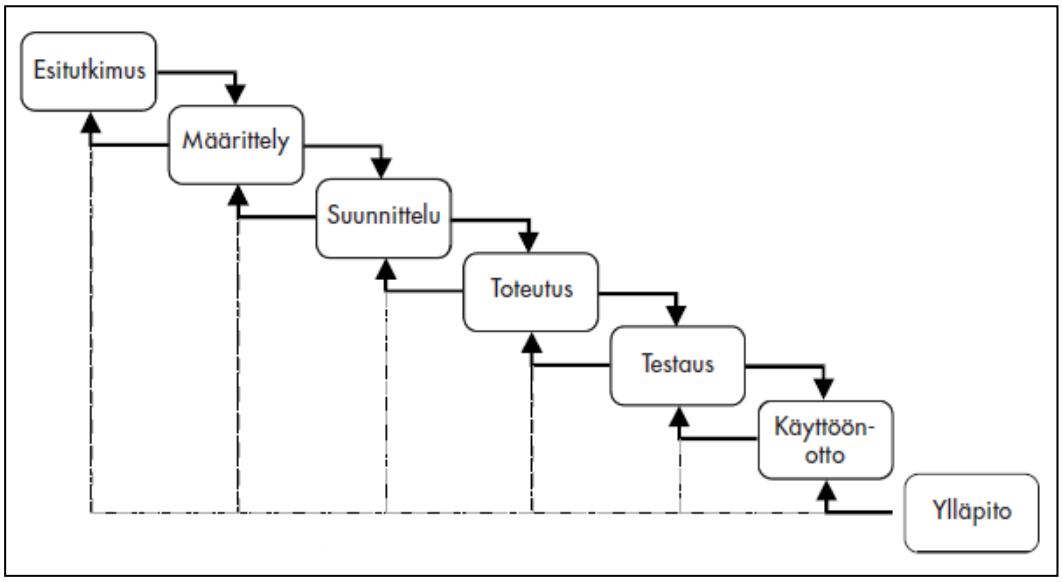
Kehittäjien ja asiakkaan välinen yhteistyö. Kehittäjien ja asiakkaan välistä yhteistyötä pidetään tärkeämpänä kuin tarkkoja sopimuksia, joskin hyvin tehtyjen sopimuksien tärkeys kasvaa samassa suhteessa kuin ohjelmistoprojektikin. Neuvotteluprosessi tulee nähdä keinona saavuttaa ja ylläpitää asiakassuhde. Liiketoiminnan näkökulmasta katsottuna ketterä kehitys keskittyy heti projektin alussa luomaan arvoa liiketoiminnalle, mikä vähentää riskejä toteuttamisen osalta. Tehokkain tapa välittää tietoa

kehitystiimin ja asiakkaan välillä on kasvotusten tapahtuva keskustelu. (Abrahamsson ym. 2002, 11–12; Beck ym. 2001.)

Reagoiminen muutoksiin. Kehitysryhmän jäsenten, joita ovat sekä sovelluksen kehittäjät että asiakkaan edustajat, tulee olla tietoisia, teknisesti osaavia ja valtuutettuja tekemään tarvittavia muutoksia kehitysprosessin elinkaaren aikana. Kesken projektia tulevat muutosehdotukset tulee ottaa huomioon myös projektin loppuvaiheessa, jotta asiakas saa parhaan mahdollisen hyödyn ja kilpailuedun. Käytössä olevan sopimuksen tulee olla muotoiltu niin, että se sallii tarvittavat kehitystoimet. (Abrahamsson ym. 2002, 11–12; Beck ym. 2001.)

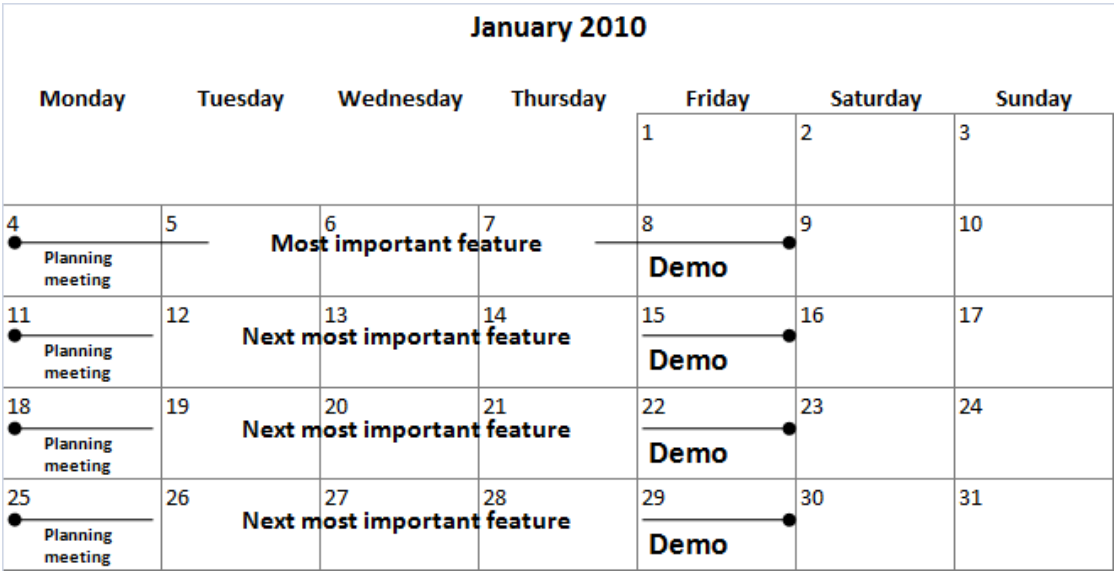
3.2 Perinteiset vs. ketterät menetelmät

Perinteiset tuotekehitysmallit on suunniteltu teknisten ja fyysisten tuotteiden tuotekehitykseen sekä teollisen tuotannon tarpeisiin (Manninen 2009, 9). Perinteisin malli on vesiputousmalli, jonka periaatteena on eri vaiheiden perättäisyys (ks.kuvio alla). Vesiputousmallissa vaiheet seuraavat toisiaan suoraviivaisesti esitutkimuksesta alkaen ja päättyen ylläpitoon. Ideaalitilanteessa ohjelmistoprosessi menee niin kuin on suunniteltu, mutta todellisuudessa harva kehityshanke pystyy seuraamaan tätä lineaarista kaavaa. Perinteisten menetelmien ongelmana nähdään selkeästi niiden kyvyttömyys reagoida muutoksiin ajoissa tai välttämättä ollenkaan, sillä muutosten tekeminen vaatii usein useamman vaiheen uusimista. Tämä on kallista asiakkaalle ja työlästä kehitysryhmälle. (Pohjonen 2002, 40.)



KUVIO 1. Vesiputousmallin elinkaari (Pohjonen 2002, 40)

Ketterien menetelmien suurimpia vahvuuksia taas ovat reagoimiskyky silloin, kun muutoksia täytyy tehdä nopeasti. Ketterät menetelmät tuottavat asiakkaalle jatkuvasti, lyhyin väliajoin toimivan version tuotteesta, ja asiakas on tiiviissä yhteistyössä kehittäjien kanssa (ks. kuvio alla). Kuviosta 2 voidaan havaita, että jokainen viikko on pieni ohjelmistoprojekti, jonka lopputuloksena on toimiva tuote (demo). Jokainen viikko sisältää asiakkaan vaatimusten mukaisen dokumentoinnin, suunnittelun, koodauksen ja testauksen. Tämä takaa sen, että asiakas tietää, missä projekti menee, pystyy antamaan palautetta sekä tekemään tarvittavia muutoksia projektin aikana. (Wells 2009c.)



KUVIO 2. XP:n prosessi kalenterinäkymässä (Wells 2009c)

Toinen merkittävä eroavaisuus löytyy dokumentoinnista. Perinteisissä menetelmissä kattavaan dokumentaatioon käytetään paljon aikaa. Ketterissä menetelmissä taas ensisijalla on kuitenkin itse lopputulos, eikä dokumentaation suhteen aseteta tiukkoja vaatimuksia. Perussääntönä on, että dokumentoidaan vain tarvittava osa sovellusta tai sen kehittämistä, jolloin vastuu jää täysin kehitystiimille. Tämä saattaa synnyttää harhakuvan siitä, että ketterä ohjelmistokehitys on kuritonta ja suunnittelematonta kehitystä. Näin ei kuitenkaan ole, sillä suunnitelmia tehdään aina tarpeen vaatiessa ja samalla ollaan valmiita reagoimaan muutoksiin nopeasti ja tehokkaasti.

3.3 Scrum

Scrum on projektihallinnan menetelmä, joka tarjoaa arvoja ja sääntöjä sovelluskehitysprojektien ohjaamiseen. Se ei niinkään ota kantaa eri käytäntöihin vaan keskittyy projektin vaiheistamiseen ja kontrollointiin (Ketterät käytännöt n.d.). Scrumia on käytetty menestyksekkäästi tuhansissa projekteissa sadoissa organisaatioissa viimeisten kuudentoista vuoden aikana. Scrum sopii projekteihin, joissa ymmärretään, että hankalat ongelmat voidaan voittaa päättäväisyydellä ja kekseliäisyydellä. (Schwaber 2007.)

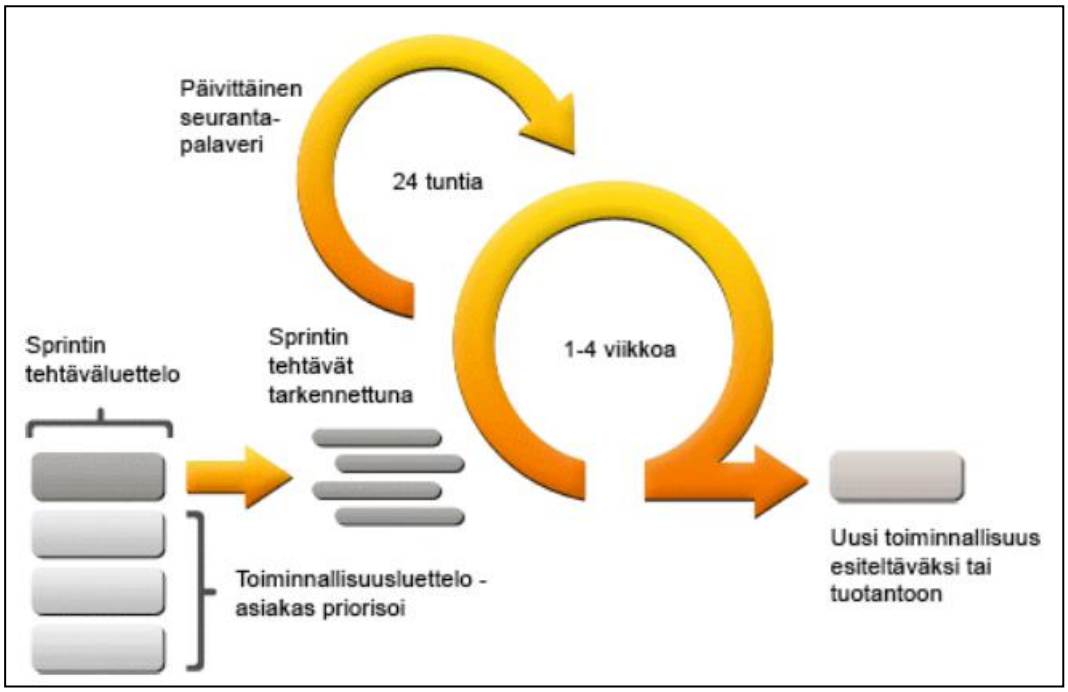
Scrum-rooleja on kolme: tuotteen omistaja (Product Owner), Scrum-tiimi (Scrum Team) ja Scrum-mestari (Scrum Master). Kaikki projektin hallinnolliset vastuut jaetaan näiden roolien kesken. Schwaber (2007) kuvaa Mannisen (2009) mukaan kirjassaan *Agile Project Management* (2004) sekä artikkelissaan *What is Scrum? (2007)* Scrum-rooleja seuraavasti:

Tuotteen omistaja, eli asiakas, on vastuussa siitä, että lopputuotteen vaatimukset ja niiden tärkeysjärjestys ovat selvillä koko ajan ja koko tiimille. Nämä vaatimukset kirjoitetaan työlistaan (Product Backlog). Asiakkaan vastuulla on huolehtia siitä, että kaikista tärkeimmät vaatimukset menevät tuotantoon ensin. Tästä johtuen asiakkaan tulee päivittää työlistaa säännöllisesti sitä mukaa, kun vaatimuksia toteutetaan.

Scrum-tiimin vastuulla on tuottaa asiakkaan haluamana toiminnallisuus työlistan vaatimusten pohjalta itsenäisesti ja tasa-arvoisesti. Jokaisen iteraation onnistumisesta vastaa yksinomaan tiimi. Tämän vuoksi tiimin kehittyminen ylhäältä johdetusta tiimistä täysin itseohjautuvaksi on välttämätöntä projektin onnistumisen kannalta.

Scrum-mestarin rooli projektissa taas on huolehtia prosessin toimivuudesta ja siitä, että se etenee Scrumin käytäntöjen, arvojen ja sääntöjen mukaan muokkaamalla ja opettamalla sitä eteenpäin projektissa mukana oleville henkilöille niin, että se prosessi soveltuu mahdollisimman hyvin kyseiseen organisaatioon. Scrum-mestari toimii Scrum-tiimin ja asiakkaan yhteyshenkilönä. Sakaryania (2010) lainaten Scrum-mestarin tehtäviin kuuluu tämän lisäksi muun muassa yleinen asioiden organisointi, projektin aikana ilmenevien ongelmien ratkaisu sekä sprinttien loppuun viennin auttaminen. Sakaryanian (2010) mukaan Scrum-mestarin rooli on todella haastava ja hänen mielestään tehtävään sopii henkilö, jolla on hyvät johtamis- ja kommunikointitaidot sekä malttia. Tekninen osaaminen ei ole tässä tapauksessa tärkeässä osassa.

Ketterien menetelmien mukaisesti myös Scrumin prosessi rakentuu erimittaisten syklien ympärille. Tärkeimmät syklit ovat sprintti ja päivä. (Ketterät käytännöt n.d.) Scrumin prosessi on kuvattu alla.



KUVIO 3. Scrumin prosessi (Koskela n.d.)

Scrum-projekti aloitetaan luomalla visio rakennettavasta järjestelmästä, suunnittelemalla aikataulutusta ja kustannuksia sekä tekemällä yksinkertainen työlista, joka sisältää kaikki vaatimukset järjestelmää koskien. Aluksi vision hahmottaminen voi olla vaikeaa ja siitä voi tulla epämääräinen. Tämän vuoksi sitä voidaan tarkentaa projektin edetessä. Kuten todettua, asiakas on vastuussa tekemästään työlistasta, joka sisältää järjestelmän toiminnalliset ja ei-toiminnalliset vaatimukset ja niiden tärkeysjärjestyksen, sekä siitä, että tiimi ymmärtää ne. Työlistan perusteella tehdään julkaisusuunnitelma, joka hyvin usein muuttuu projektin edetessä. (Schwaber 2007.)

Projektin varsinainen työ tehdään sprinteissä. Jokainen sprintti kestää normaalisti yhden kuukauden, mutta sen kesto voi vaihdella projektista riippuen myös viikosta kahteen kuukauteen. (Ketterät käytännöt n.d.) Jokainen sprintti aloitetaan aloituspalaverilla, jossa ovat mukana asiakas sekä Scrum-tiimi. Asiakas ja tiimi päättävät yhdessä, mikä on seuraavan sprintin tuotos. Käytännössä tämä voidaan toteuttaa niin, että ensin asiakas esittelee tiimille vaatimuksia työlistalta niiden tärkeysjärjestyksen perusteella, ja tiimi voi esittää tarkentavia kysymyksiä. Kun tiimillä on tarpeeksi hyvä kuva siitä, mitä asiakas haluaa toteutettavan, sekä siitä, kuinka paljon mikäkin vaati-

mus tulee viemään aikaa, tiimi valitsee ne vaatimukset työlistalta, jotka se uskoo pystyvänsä toteuttamaan seuraavan sprintin aikana. Tämän jälkeen tiimi suunnittelee sprintin toteutuksen. Scrumin mukaisesti tiimin tulee sprintin aikana toteuttaa inkrementti, jonka asiakas voi halutessaan heti lisätä tuotteeseen. Inkrementin tulee olla testattu ja käyttäjäoperaatioiden tulee olla dokumentoituina. Koska tiimi on itse vastuussa sprintin onnistumisesta, on sen hyvä määritellä pikemminkin varovainen suunnitelma kuin yrittää toteuttaa liian montaa vaatimusta yhdessä sprintissä. (Schwaber 2007.)

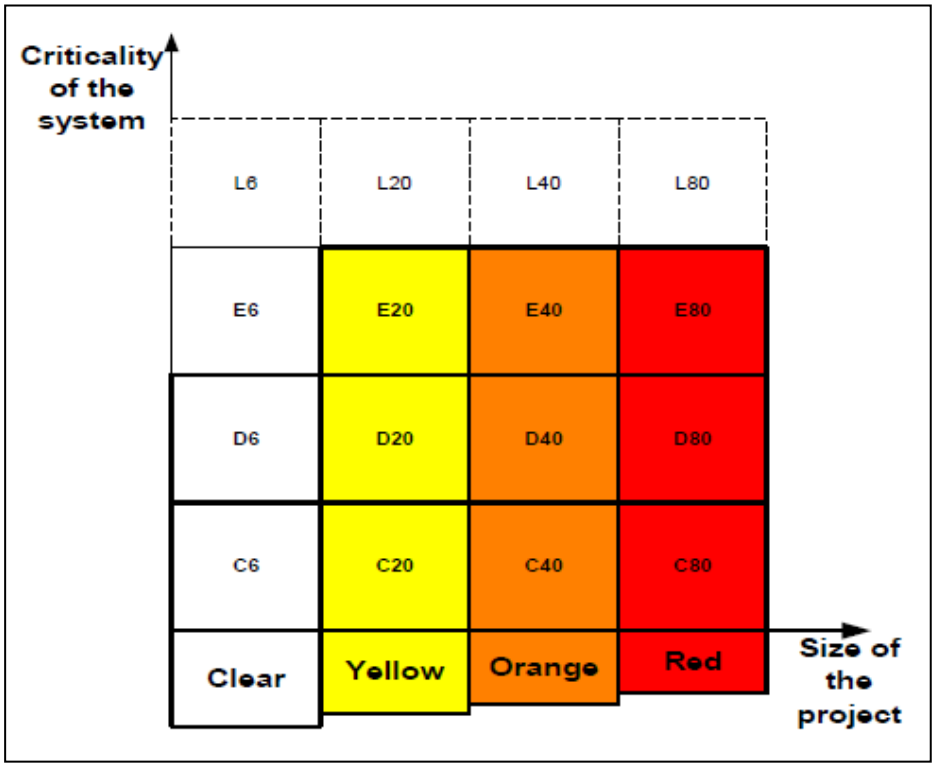
Toinen Scrumin käyttämä sykli on päivä. Jokainen päivä aloitetaan 15 minuutin palaverilla, jossa tiimin jäsenet kertovat vuorollaan, mitä ovat tehneet viimeisimmän palaverin jälkeen, mitä aikovat tehdä seuraavaksi sekä kertovat mahdollisista havaituista ongelmista. Palaverin tarkoituksena on jakaa ja tasata tiimin jäsenten työmäärää päivittäin sekä huolehtia siitä, että tiimi on pyrkimässä kohti päämääräänsä. (Schwaber 2007.)

Sprintin lopussa pidetään katselmus, jossa tiimin jäsenet esittelevät sprintin aikana tehdyn tuotoksen asiakkaalle ja mahdollisesti myös muille osakkaille, jotka haluavat osallistua kokoukseen. Palaverin pohjalta asiakas pysyy perillä siitä, mitä on tehty, millä tavalla ja mikä tulee olemaan seuraava inkrementti. Ennen seuraavan sprintin aloituspalaveria Scrum-mestari pitää vielä palaverin tiimilleen. Tässä palaverissa Scrum-mestarin tulee rohkaista tiimiään arvioimaan omaa työtään, jotta sitä pystytään kehittämään seuraavaa sprinttiä silmälläpitäen. (Schwaber 2007.)

3.4 Crystal

Crystal-menetelmät sisältävät muutamia erilaisia metodologioita, joista voidaan valita sopivin yksittäiseen projektiin. Crystal-menetelmässä esiin tuleva käsite on raskaus (Heaviness). Jokaista metodologiaa kuvastaa väri; mitä tummempi väri sen raskaampi metodologia. Crystal-menetelmän mukaisesti oikea metodologia tulisi valita väreistä sen mukaan, miten suuri ja kriittinen projekti on kyseessä. Kriittisyys luokitellaan neljään luokkaan: Comfort (mukavuus, C), Discretionary money (harkinnanvarainen ra-

ha, D), Essential money (välttämätön raha, E) ja Life (kesto, L). Projektin koko taas määritellään toteuttajamäärän mukaan.



KUVIO 4. Crystal-menetelmän raskaus projektikoon ja kriittisyyden mukaan (Abrahamsson ym. 2002, 37)

Kaikilla Crystal-menetelmillä on samoja sääntöjä, piirteitä ja arvoja. Näitä ovat Kettusen (2008, 32) mukaan inkrementaaliset kehityssyklit (maksimissaan neljä kuukautta) ja ihmisten välisen kanssakäynnin ja yhteistyön korostaminen. Ominaista Crystal-menetelmille on myös se, etteivät ne automaattisesti sulje pois mitään tiettyä kehityskäytäntöä, vaan hyväksyvät, että projektin muuttuessa käytäntöjä voidaan mukauttaa tarpeen vaatiessa. Crystal Clear on suunniteltu pieniin projekteihin, joissa on maksimissaan kahdeksan kehittäjää, jotka kaikki työskentelevät samassa paikassa. Crystal Clear sopii esimerkiksi projekteihin, jotka eivät pysty toimimaan kurinalaisemman XP:n alla. Crystal Clear antaa tiimille enemmän vapauksia määrittää, mitä ketteriä menetelmiä ja tekniikoita he tiimissään haluavat käyttää, kunhan ne sopivat yleisesti metodologiaan. Crystal Orange taas on suunniteltu käytettäväksi projekteis-

sa, joissa raha ja nopea tuottavuus ovat tärkeitä ja projektihenkilöstön määrä on 20–50 henkilöä (Schuh 2004, 32–33).

Abrahamsson ja muut (2002, 46) arvostelevat Crystal-menetelmiä niiden kommunikaatiota koskevien rajoitusten vuoksi. Esimerkiksi Crystal Clear -menetelmässä kommunikointi on niin rajoitettua, että se soveltuu vain samassa työtilassa työskentelevälle yksittäiselle ryhmälle. Crystal Orange vaatii, että kehitysryhmien tulee sijaita samassa rakennuksessa. Näin ollen Crystal-menetelmät eivät sovellu turvallisuuskriittisten (Life-critical) sovellusten kehittämiseen.

4 EXTREME PROGRAMMING (XP)

Tässä luvussa esitellään ketteristä menetelmistä Extreme Programming (XP). Luvussa perehdytään XP:n perusarvoihin, elinkaareen, rooleihin sekä varsinaiseen prosessiin. Luvun lopussa tutustutaan lisäksi kahteen XP-projektiin ja niistä saatuihin kokemuksiin.

4.1 Esittely

Yhden XP:n kehittäjistä, Beck Kentin (2000, xiii, 3–5) mukaan XP on kevyt, tehokas, joustava ja pieniriskinen menetelmä, joka sopii pienille ja keskisuurille ohjelmistokehitystiimeille. XP tarjoaa arvoja ja menetelmiä, joiden avulla ohjelmoijat pystyvät tekemään sen, minkä he osaavat parhaiten: tuottamaan koodia. XP:n vahvuuksia ovat reagoimiskyky muutoksiin säännöllisten ja lyhyiden iteraatioiden ansiosta, testatun ja toimivan ohjelmiston toimittaminen asiakkaalle pienissä erissä sekä tiivis asiakasyhteistyö.

Lindbergin (2003, 40) mukaan Beck (1999) kuvaa XP:n merkitystä avaamalla nimen molempia sanoja. Sanalla Extreme (äärimmäisyys) Beck (1999) tarkoittaa, että XP:ssä hyväksi havaitut käytännöt viedään äärimmäisyyksiin. Tästä esimerkkinä pariohjelmointi, joka on katselmoinnin viemistä äärimmäisyyksiin, jatkuva testaus, joka on testauksen viemistä äärimmäisyyksiin, ja nopea evoluutiosykli, joka on palautteen viemistä äärimmäisyyksiin. Sanalla Programming (ohjelmointi) Beck (1999) tarkoittaa XP:n olevan nimenomaisesti ohjelmointiin painottunut menetelmä, jossa dokumentointi ei ole yhtä tärkeää kuin monissa muissa menetelmissä. Pääasia on toteuttaa mahdollisimman yksinkertaisia ohjelmia vähäisellä suunnittelulla ja dokumentoinnilla, jolloin aikaa jää varsinaiseen ohjelmointiin. Suunnittelun puutteesta aiheutuvia huonoja arkkitehtuuriratkaisuja pyritään korjaamaan koodia refaktoroimalla.

XP pohjautuu viiteen arvoon, joiden kautta käytäntöjen ymmärtäminen on helpompaa. Näitä arvoja ovat kommunikointi, yksinkertaisuus, palaute, kunnioitus ja rohkeus. XP:n perusarvot on esitelty tarkemmin seuraavassa luvussa.

4.2 Perusarvot

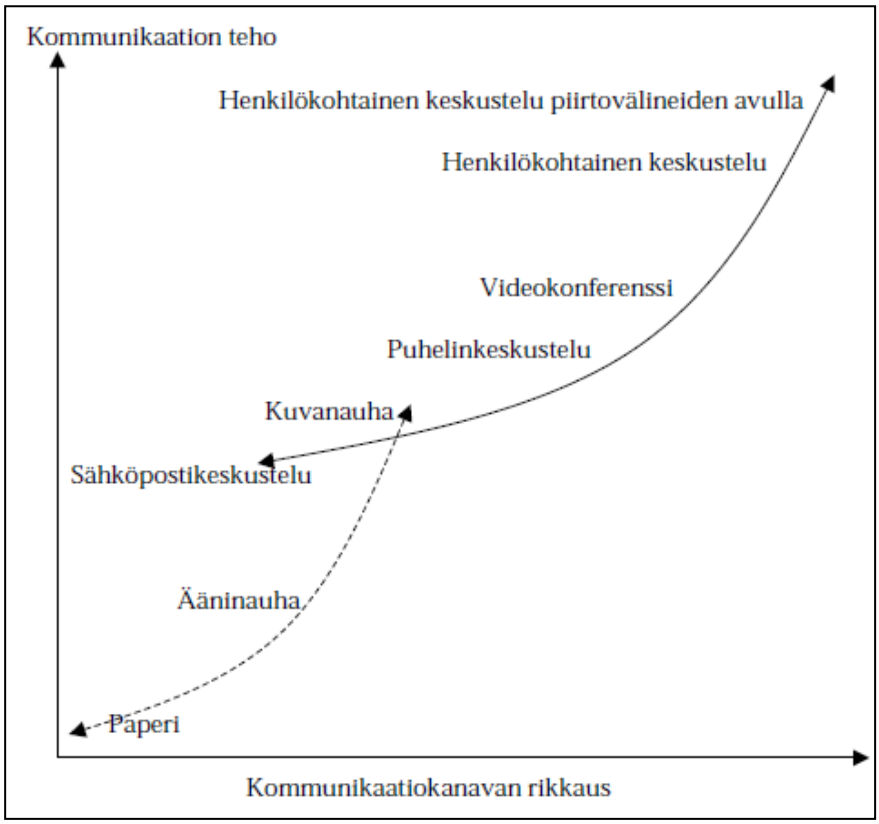
XP perustuu viiteen perusarvoon: yksinkertaisuuteen, kommunikointiin, palautteeseen, kunnioitukseen ja rohkeuteen. Vaikka XP voidaan ymmärtää paljon tiukkoja sääntöjä antavana menetelmänä, sen perimmäinen tarkoitus on kuitenkin tarjota työskentelytapoja, jotka helpottavat päivittäistä työtä. (Wells 2009b.)

Yksinkertaisuus

Yksinkertaisuudella tarkoitetaan sitä, että järjestelmästä tehdään juuri sellainen kuin asiakas haluaa, eikä mitään ylimääräistä toteuteta (Wells 2009b). Järjestelmää kehitetään lyhyissä iteraatioissa, mikä yhdessä tiiviin asiakasyhteystyön kanssa takaa sen, että asiakas on koko ajan tietoinen projektin etenemisestä ja voi puuttua havaitsemiinsa ongelmiin hyvissä ajoin. Lindbergin (2003, 16) mukaan suunnittelussa ei saisi koskaan varautua sellaisiin vaatimuksiin, joita ei suunnitteluhetkellä ole tiedossa. Näin järjestelmää ei tietoisesti laajenneta eri suuntiin. Siitä, aiheuttaako tämä enemmän lisätyötä myöhemmin, voidaan tapauskohtaisesti olla montaa mieltä. XP:n arvojen mukaisesti iteraatiossa toteutetaan kuitenkin aina vain se osa järjestelmää, jonka vaatimukset iteraatioon on määritelty.

Kommunikointi

Lindbergin (2003, 51–52) mukaan Cockburn (2002) esittää alla olevan kuvion mukaisen mallin kommunikointikanavien tehokkuudesta eri välineitä käytettäessä. Kuviossa on esitetty katkoviivalla eri dokumentaatiovälineitä ja yhtenäisellä viivalla erilaisia välineitä ohjelmiston mallintamiseen. Paperidokumentaatio nähdään heikoimmaksi dokumentaatiovälineeksi, vaikka perinteisesti sitä onkin käytetty ohjelmistoprojekteissa paljon. Liikkuva kuva ja ääni tuovat dokumentaatioon uusia mahdollisuuksia, kun taas paperidokumentaatio rajoittaa ilmaisun kuviin ja tekstiin. Kommunikaation teho paranee sitä mukaa, kuin ilmaisuvoima kasvaa. Näin ollen tehokkaimmaksi kommunikaatiokanavaksi nähdään henkilökohtainen keskustelu piirtovälineiden, esimerkiksi fläppitaulun, avulla. Eri kanavien tehokkuuteen saattavat kuitenkin vaikuttaa muutkin seikat, kuten viestin lähettäjä, sen vastaanottaja ja viestin sisältö.



KUVIO 5. Erilaisten kommunikointikanavien tehokkuus (Lindberg 2003, 51)

Paras tilanne Lindbergin (2003, 52) mukaan olisi sellainen, jossa järjestelmää voitaisiin kehittää asiakkaan läsnä ollessa. Lindbergin omat kokemukset sekä lähteet ovat kuitenkin osoittaneet, että asiakkaalla on useimmiten kiire omien töidensä kanssa, eikä jatkuva läsnäolo ole tällöin mitenkään mahdollista. Projektiryhmän kannalta asiakkaan jatkuva läsnäolo olisi ihanteellinen tilanne, mutta asiakkaalle siitä koituisi usein kustannusten kasvua. Asiakkaan ja projektiryhmän välisen kommunikoinnin lisäksi avainasemassa on projektiryhmän sisäinen kommunikaatio. Peltoniemen (2009, 61) mukaan XP:ssä noudatetaan työtapoja, joita on mahdotonta toteuttaa ilman ihmisten välistä kommunikaatiota. On esimerkiksi ilmiselvää, ettei pariohjelmointia voida tehokkaasti toteuttaa, ellei ohjelmoijien yhteishenki ole hyvä ja kommunikaatio ole samalla tasolla. Kommunikaation puute on toki vaarana myös XP-projekteissa, sillä dokumentaatiota ei tehdä yhtä paljon kuin muissa menetelmissä. XP:n perusidea onkin luottaa enemmän kasvotusten tapahtuvaan keskusteluun dokumentoinnin sijasta.

Palaute

Palautteen antaminen ja kerääminen on ensisijainen keino saada järjestelmästä sellainen, kuin asiakas haluaa. Asiakkaalta pyritään saamaan palautetta mahdollisimman usein, viimeistään säännöllisesti iteraatioiden jälkeen. Lyhyet iteraatiot ja tiivis asiakasyhteistyö pyrkivätkin takaamaan sen, että palautetta annetaan ja saadaan säännöllisesti. (Wells 2009b.)

Kunnioitus

XP-projektin tiimin jokaisen jäsenen tulee tuntea olonsa tarpeelliseksi sekä antaa tunnustusta myös muille heidän työstään. Kehittäjät kunnioittavat asiakkaan osaamista ja päinvastoin. (Wells 2009b.)

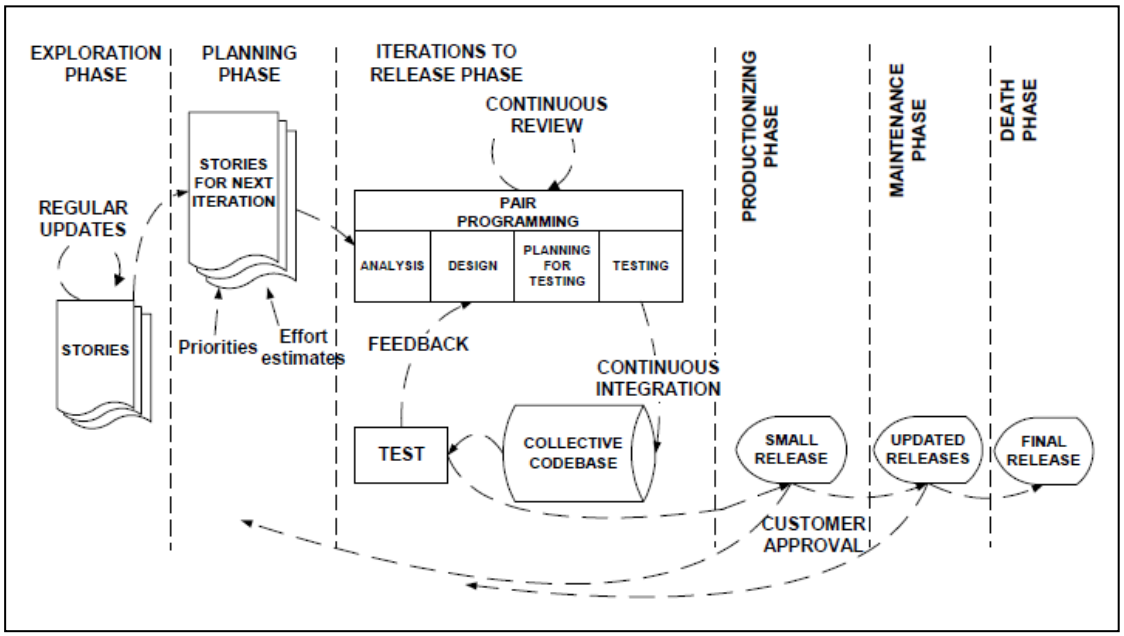
Rohkeus

Voidaan ajatella, että jo yksinomaan XP:n valitseminen käytettäväksi menetelmäksi vaatii rohkeutta sen asettamien sääntöjen ja arvojen puolesta. Tämä johtuu esimerkiksi siitä, ettei XP:ssä tehdä tarkkoja dokumentteja työn etenemisestä tai järjestelmästä, vaan sen annetaan ikään kuin valmistua omalla painollaan toisin kuin muissa menetelmissä, joissa pyritään varautumaan tulevaisuuteen luotujen dokumenttien pohjalta. Peltoniemen (2009, 62–63) mukaan XP:ssä rohkeudella tarkoitetaan sitä, että suunnittelutyö uskalletaan lopettaa ajoissa ja alkaa tuottaa varsinaista tuotetta, vaikka kaikkia tulevaisuudessa mahdollisesti esiin tulevia ongelmia ei olekaan voitu vielä selvittää.

4.3 Elinkaari

XP:n elinkaari koostuu kuudesta vaiheesta: tutkiminen, suunnitteleminen, julkaistavien iteraatioiden määritys, tuotteistaminen, ylläpito ja päättäminen (ks.kuvio alla). Yksinkertaisimmillaan XP:n elinkaari voidaan jakaa kolmeen osaan: tutkimus, toteutus ja lopullinen tuote. Toteutusvaihe, eli iteraatio, toistuu projektin aikana niin kauan, kunnes asiakkaan tavoitteet on saavutettu tai projekti päätetään muusta syystä viedä loppuun. Toteutusvaihe pitää sisällään suunnittelun, määritellyn, toteutuksen, testauksen ja julkaisun. Toteutusvaiheiden lopputuloksista koostuu lopullinen tuote

asiakkaalle. (Abrahamsson ym. 2002, 19.) Tällä tavoin XP rakentuu kerroksittain lopulliseksi tuotteeksi. Jokainen iteraatio pitää sisällään erilaisia menetelmiä, jotka ovat tuttuja jo entuudestaan ohjelmistotuotannosta. XP:ssä nämä menetelmät on koottu yhteen uudella innovatiivisella tavalla. Tässä luvussa perehdytään XP:n elinkaaren eri vaiheisiin sekä niiden sisältämiin menetelmiin ja käytäntöihin.



KUVIO 6. XP:n elinkaari (Abrahamsson ym. 2002, 19)

4.3.1 Tutkiminen

Tutkimusvaihe tulee suorittaa mahdollisimman nopeasti, sillä projektiryhmän pitäminen poissa tuotantovaiheesta tarkoittaa rahan tuhlaamista. XP:ssä ei oteta kovin tiiviisti kantaa tutkimusvaiheeseen, mutta Beck (2000, 133) toteaa kuitenkin, että tutkimusvaihe voidaan sisällyttää projektin alkuun ja se on hyvä toteuttaa etenkin silloin, kun käytettävä teknologia ei ole projektiryhmälle entuudestaan tuttu. Tällöin tutkimusvaiheessa tutustutaan teknologiaan, harjoitellaan sen käyttöä ja valmistellaan projektiryhmää tulevan työn toteuttamiseen.

Tutkimusvaiheessa asiakas aloittaa käyttäjätarinoiden (User Stories) kirjoittamisen ensimmäistä julkaisua varten. Käyttäjätarinat ovat eräänlaisia käyttötapauksia (Use

Case). Niissä kuvataan järjestelmän vaatimuksia, muttei kuitenkaan teknisiä yksityiskohtia. Kun asiakas on kirjoittanut käyttäjätarinat, kehitysryhmä lukee ne ja antaa niistä palautetta. Tätä kautta yksi XP:n perusarvoista, palautteen anto ja vastaanotto, tulee mukaan prosessiin. Palautteen kautta käyttäjätarinoita voidaan muokata paremmiksi. Ohjelmointi voidaan aloittaa sitten, kun käyttäjätarinat ovat riittävän yksityiskohtaisia toteutuksen näkökulmasta, ja kun kehittäjät ovat varmoja siitä, että he hallitsevat käytettävän tekniikan. Tutkimusvaihe kestää yleensä muutamasta viikosta muutamaan kuukauteen riippuen pitkälti siitä, kuinka hyvin ohjelmoijat hallitsevat käytettävän teknologian, sekä siitä, milloin asiakas on sitä mieltä, että ensimmäiseen julkaisuun on tarpeeksi käyttäjätarinoita.

4.3.2 Suunnittelu

Varsinainen iteraation suunnittelu aloitetaan määrittelyllä, jota XP:ssä kutsutaan nimellä suunnittelupalaveri (Planning Meeting) tai suunnittelupeli (Planning Game). Tässä työssä tullaan käyttämään käsitettä suunnittelupalaveri. Suunnittelupalaverin tarkoituksena on päättää julkaisun sisältö ja aikataulu. Suunnittelupalaveri toteutetaan käytännössä siten, että asiakas esittää järjestelmän vaatimuksia ja priorisoi niitä. Tämän jälkeen kehitysryhmä antaa jokaisesta vaatimuksesta työmäärä-, aikataulu- ja kustannusarvion. Näiden tietojen perusteella asiakas päättää, mitä toimintoja seuraavaan iteraatioon tulee. Ne toiminnot, joita ei seuraavassa iteraatiossa toteuteta, säästetään seuraavaa suunnittelupalaveria varten. (Lindberg 2003, 17.)

Aikataulu määritellään yhdessä asiakkaan ja kehitysryhmän kanssa kehittäjien työmääräarvioiden ja asiakkaan priorisoinnin pohjalta. Yksittäinen iteraatio kestää normaalisti yhdestä neljään viikkoa. Ensimmäisen iteraation on määrä esittää koko järjestelmän arkkitehtuuri, joten ensimmäisen julkaisun suunnittelussa tulee keskyttyä valitsemaan tähän sopivia käyttäjätarinoita. Ensimmäisen julkaisun aikataulu on yleensä kahdesta viiteen kuukautta. Asiakas myös tekee käytännön testit, jotka suoritetaan jokaisen iteraation jälkeen asiakkaan toimesta. Viimeisen iteraation valmistumisen jälkeen järjestelmä voidaan siirtää tuotantoon. (Abrahamsson ym. 2002, 20; Lindberg 2003, 17, 40–42.)

Koska suunnittelupalaveri pidetään ennen jokaista iteraatiota, se antaa asiakkaalle myös mahdollisuuden muuttaa vaatimuksia projektin aikana sekä vaihtaa eri iteraatioiden sisältöä. Suunnittelupalaveri on eräänlainen sovellettu yksinkertainen käyttötapaustekniikka, ja sen avulla voidaan helpottaa ohjelmiston määrittämistä sellaisten asiakkaiden kanssa, joilla ei ole aikaisempaa kokemusta ohjelmistoprojekteista. Suunnitteluvaihe kestää muutaman päivän. (Lindberg 2003, 17, 40–42.)

Suunnitelupalaverin aikana projektiryhmä ja asiakas sopivat myös kielellisestä **metaforasta**, jonka avulla järjestelmän toimintaa mallinnetaan. Metaforalla tarkoitetaan jonkin tunnetun käsitteen käyttöä auttamaan vähemmän tunnetun käsitteen ymmärtämistä. Metaforan tarkoitus on selventää teknisiä asioita ja pitää kommunikaatio mahdollisimman yksinkertaisena ja sujuvana. Esimerkiksi verkkokaupan metafora voisi olla valintamyymälä. Näillä kahdella on paljon samoja termejä, kuten ostoskori, osasto ja kassa. (Lindberg 2003, 17–18.)

4.3.3 Toteutus

XP:n eroavaisuudet verrattuna muihin menetelmiin tulevat voimakkaimmin esille, kun siirrytään toteutusvaiheeseen. Ennen kuin ohjelmointi aloitetaan, ohjelmoijat kirjoittavat käyttäjätarinoille moduulitestitapauksen (Test Case), joka yleensä integroidaan automatisoituun testausjärjestelmään. Vasta tämän jälkeen aloitetaan varsinaisen ohjelmakoodin kirjoittaminen. Tätä tekniikkaa kutsutaan **testilähtöiseksi kehittämiseksi** (Test-driven Development). Ohjelmakoodi kirjoitetaan niin, että se toteuttaa käyttäjätarinan, muttei mitään muuta. Toisin sanoen pyritään etsimään yksinkertaisin mahdollinen ratkaisu vaatimuksen toteuttamiseen. Jatkossa tuleviin vaatimuksiin ei tässä vaiheessa oteta kantaa, vaan ne jätetään rohkeasti odottamaan vuoroaan. Kun vaatimus on ohjelmoitu, ajetaan kaikki testausjärjestelmän sisältämät testit. Mikäli testit menevät läpi, on vaihe onnistunut ja voidaan siirtyä seuraavan vaatimuksen testin kirjoittamiseen. Jos testausjärjestelmä palauttaa virheen, virhe korjataan ja testit ajetaan uudestaan niin monta kertaa, kunnes ne menevät läpi. (Lindberg 2003, 18–19.)

XP:ssä kaikki ohjelmakoodi tuotetaan **pariohjelmointina**. Tämä tarkoittaa käytännössä sitä, että muodostetaan pareja, joista toinen kirjoittaa koodia ja toinen seuraa ja ilmoittaa havaitsemistaan virheistä. Ohjelmoijat keskustelevat myös keskenään parhaista toteutusvaihtoehdoista. Pariohjelmoinnissa rooleja voidaan vaihtaa tarpeen mukaan, esimerkiksi silloin, kun koodia kirjoittava henkilö ei osaa toteuttaa toisen ehdottamaa rakennetta. Perussääntönä pidetään sitä, että molemmat ohjelmoijat toimivat vuorollaan sekä kirjoittajina että seuraajina. Pariohjelmointi johtaa osaltaan siihen, että kirjoitettu koodi on ohjelmoijien **yhteisomistuksessa**. Tällöin kenen tahansa ohjelmoijista tulisi käytännössä pystyä muokkaamaan mitä tahansa osaa koodista. Koodin yhteisomistus vaatii kuitenkin ohjelmointityylin standardoinnin määrittelemään koodin ulkoasua ja teknistä rakennetta. (Lindberg 2003, 19.) Tällä standardoinnilla tarkoitetaan koodauskonvention käyttöä.

Lindbergin (2003, 50) mukaan **koodauskonventio** koostuu nimeämistyylistä ja ohjelmointityylistä. Koodauskonventiota voidaan kutsua myös **koodistandardiksi**. Nimeämistyylillä tarkoitetaan esimerkiksi sitä, miten ohjelmoija nimeää käyttämänsä muuttujat, metodit ja luokat. Ohjelmointityylillä tarkoitetaan taas sitä, miltä ohjelmakoodi näyttää visuaalisesti, eli miten se on sisennetty ja kuinka ohjelman eri osat on erotettu toisistaan. Huonosti nimetyt muuttujat ja vaihteleva ohjelmointityyli hidastavat koodin ymmärtämistä, vievät sitä kautta aikaa ja johtavat väärinymmärryksiin ja ohjelmointivirheisiin. Käytetyistä koodauskonventioista voidaan sopia esimerkiksi projektiryhmäkohtaisesti.

Koodia saa muokata vain silloin, kun testit eivät mene läpi, tai refaktoroimalla. **Refaktoroinnilla** tarkoitetaan ohjelmakoodin arkkitehtuurin muuttamista ilman sen toiminnallisuuden muuttamista. Virheiden korjausta tai nimeämiskäytännön muutoksia ei näin ollen voida pitää koodin refaktorointina. (Lindberg 2003, 19, 45.) Tavallisesti ohjelmoijille on ollut tyypillistä rakentaa koodia ja muokata sitä vain niiltä osin, kuin se on aivan välttämätöntä. Tämä johtaa siihen, että käytetään koodia, joka on huonosti ylläpidettävää. Koska koodi toimii jotenkuten, ohjelmoija ei uskalla mennä tekemään siihen suuria muutoksia siinä pelossa, että koodi lakkaa toimimasta. XP:n

refaktorointikäytäntö pitää huolen siitä, että koodista karsitaan ylimääräinen pois ja se pysyy näin ollen yksinkertaisena ja tarkoituksenmukaisena. (Wells 2009a.)

Koodin pitäminen **yksinkertaisena** takaa sen, että sitä on helppo ymmärtää, muokata ja laajentaa. Aluksi tämä voi tuntua vaikealta, sillä oikean mallin löytäminen tietyn ominaisuuden toteuttamiseen ei aina onnistu ensimmäisellä kerralla. Tärkeää on muistaa, että vaikka jokin ominaisuus toimisikin ensi yrittämällä, voi sen toteuttamiseksi olla olemassa useita vaihtoehtoja, joita kannattaa kokeilla. (Wells 2009a.) Koodin refaktorointi on erityisen tarpeellinen juuri XP-projekteissa, sillä niissä ei suunnitella ohjelmiston arkkitehtuuria etukäteen. Lindbergin (2003, 46) mukaan tämä on XP:n yksi eniten kritisoituja ominaisuuksia, sillä yleinen mielipide ohjelmistokehittäjien keskuudessa on, että on parempi suunnitella hyvä arkkitehtuuri projektin alussa kuin alkaa muuttaa sitä projektin aikana.

Kun kaikki iteraation vaatimusten vaatima koodi on kirjoitettu, refakoroitu ja testattu, se optimoidaan. **Optimointi** suositellaan tekemään vasta, kun kaikki koodi on kirjoitettu ja testattu, jotta optimointiin käytetty työ ei mene hukkaan, mikäli vaatimukset muuttuvat iteraation aikana. Kun koodi on moduulitestattu ja optimoitu, suoritetaan **toiminnallinen testaus** (Functional Testing). Toiminnallinen testaus on samalla myös hyväksymistestausta (Acceptance Testing). (Collins & Miller 2001.) Toiminnalliset testit on kirjoitettu ja ne suoritetaan asiakkaan toimesta projektiryhmän avustamana. Perinteisesti toiminnallinen testi liittyy käyttöliittymään ja sen antamiin tuloksiin. Mikäli virheitä havaitaan, projektiryhmä korjaa ne välittömästi ja testit suoritetaan uudestaan. Kun testit on viety onnistuneesti läpi, kyseisen iteraation ohjelmointi- ja testausvaihe päättyy. Tällöin ohjelmisto on valmis luovutettavaksi asiakkaalle sellaisenaan ja seuraavan iteraation suunnittelu voi alkaa suunnittelupalaverilla. (Lindberg 2003, 20.)

Dokumentoinnin suhteen XP määrittää, ettei mitään sellaista turhaa dokumentaatiota tehdä, josta ei ole hyötyä projektin valmistumisen jälkeen. Jonkinlainen dokumentointi voi kuitenkin olla hyödyllistä ylläpidon kannalta, eikä XP varsinaisesti kiellä dokumentointia. XP ei kuitenkaan anna ohjeita tai sääntöjä sille, minkälaisia dokument-

tien tulisi olla, käyttäjätarinoiden dokumentointia lukuun ottamatta. XP:n periaatteiden mukaisesti kaiken projektin kommunikaation tulisi tapahtua henkilökohtaisesti ihmisten välillä, ei dokumentaation kautta. (Lindberg 2003, 20.)

Collinin ja Millerin (2001) mukaan Kent Beck on sanonut haluavansa olla ”tuore ja innokas aamulla ja väsynyt ja tyytyväinen illalla”. Ylityöt kieltävä **40 tunnin työviikko** mahdollistaa tämän. Beckin mukaan työtuntien täydellistä noudattamista tärkeämpi on ymmärtää 40 tunnin työviikon periaate ja säilyttää tasainen työtahti. Vaikka kehittäjät pystyisivätkin tekemään pitkiä päiviä töissä, heidän ei sitä tule tehdä, sillä ennen pitkää he väsyvät tai saavat muita kuin työhön liittyviä ongelmia, jotka vaikuttavat työsuoritukseen. Väsyneet kehittäjät tekevät enemmän virheitä, mikä hidastaa pidemmän päälle projektia enemmän kuin niin sanotussa normaalissa aikataulussa pysyminen tekisi.

4.3.4 Tuotteistaminen

Vaikka jokaisen iteraation jälkeen tuloksena on jo osa valmista tuotetta, voidaan prosessiin laskea kuuluvaksi vielä tuotteistamisvaiheen. Tuotteistamisvaiheessa sovellusta testataan lisää ja varmistetaan sen toimivuus. Tämän jälkeen tuote voidaan julkaista. Mikäli havaitaan tarvittavia muutoksia, tehdään päätös siitä, toteutetaanko ne tässä iteraatiossa. Myöhemmäksi jätettävät muutokset ja ehdotukset kirjataan ylös ja ne toteutetaan seuraavissa iteraatioissa. (Abrahamsson ym. 2002, 20; Beck 2000, 134.)

4.3.5 Ylläpito

Ylläpitovaiheen tarkoitus on antaa tarvittaessa asiakastukea samalla, kun uusien iteraatioiden kehitys on jo käynnissä. Järjestelmän kehitys saattaa kuitenkin hidastua sen jälkeen, kun järjestelmä on tuotannossa. Tämä voi vaatia uusien työntekijöiden palkkaamista ja projektiryhmän rakenteen muuttamista. (Abrahamsson ym. 2002, 20; Beck 2000, 135–136.)

4.3.6 Päättäminen

Päätösvaihe tulee ajankohtaiseksi, kun kaikki asiakkaan kirjoittamat käyttäjätarinat on toteutettu. Tämän myötä järjestelmän tulee vastata asiakkaan tarpeita sekä olla toimiva ja luotettava. Tässä vaiheessa kirjoitetaan tarpeellinen dokumentaatio järjestelmästä, sillä sen arkkitehtuuriin, ulkoasuun tai koodiin ei enää tehdä muutoksia. Päätösvaiheeseen voidaan päätyä myös silloin, kun järjestelmä ei vastaa haluttuja tavoitteita tai jos sen jatkokehitys tulee liian kalliiksi. Päätösvaiheessa asiakas saa lopullisen, valmiin tuotteen. (Abrahamsson ym. 2002, 21; Beck 2000, 137.)

4.4 Henkilöstön roolit

XP:ssä on muutamia projektinhallintaan liittyviä rooleja siitä huolimatta, että ryhmä pyrkii tekemään päätökset yhdessä ja kommunikoimaan mahdollisimman paljon. Roolien suhteen on todettu, että ryhmä toimii paremmin, kun sen jäsenille on annettu selkeät roolit. Usein rooleja jaetaan henkilöiden kesken ja yksi henkilö voi toimia monessa roolissa. Beck (2000, 139–150) kuvaa XP:n henkilöstön rooleja kirjassaan seuraavasti:

Ohjelmoija on koko projektiryhmän peruspilari, joka suunnittelee ja kirjoittaa koodin yhdessä muiden projektiryhmän ohjelmoijien kanssa. Ohjelmoijan vastuulla on tuottaa asiakkaalle mahdollisimman hyvä tuote tämän vaatimusten pohjalta. XP:n toimintatavat vaativat ohjelmoijalta hyvää kommunikointikykyä, sillä kaikki projektiryhmän jäsenet, mukaan lukien ohjelmoijat, kommunikoivat suoraan asiakkaan kanssa, kun perinteisesti yhteyshenkilönä on toiminut projektipäällikkö. Ohjelmoijan pitää pystyä myös yksinkertaistamaan ja poimimaan asiakkaan vaatimuksista tärkeimmät kohdat. Yksinkertaistaminen heijastuu myös kirjoitetun koodin puolelle. Ohjelmoijan vastuulla on myös koodin yksikkötestaus.

Asiakas määrittelee ohjelmiston toiminnallisuutta koskevat vaatimukset käyttäjätarinoiden avulla sekä suunnittelee testit näiden pohjalta. Testien suunnittelussa apuna voivat olla myös testaajat, ja kehitysryhmän ensimmäisistä testeistä antaman palautteen perusteella asiakasta voidaan ohjata oikeaan suuntaan, mikäli hän ei ole varma,

kuinka testejä tulisi kirjoittaa. Asiakkaan vastuulla on myös tehdä projektia koskevat päätökset, valita seuraavaksi toteutettavat toiminnallisuudet, priorisoida vaatimukset ja hyväksyä toteutetut toiminnallisuudet.

Testaajia ovat käytännössä kaikki ohjelmoijat, sillä jokainen heistä on omalta osaltaan vastuussa myös testauksesta. Testaajat tarkastelevat kuitenkin myös muiden ohjelmoijien testausta, auttavat tarvittaessa, koordinoivat integrointi- ja hyväksymistestejä ja ilmoittavat esiin tulleista virheistä muille. Testaaja on viime kädessä vastuussa siitä, että haluttu toiminnallisuus on saatu toteutettua. Testaaja voi auttaa asiakasta toiminnallisten testien suunnittelussa ja myös suorittaa varsinaisen toiminnallisen testauksen.

Mittaajan (Tracker) tehtävänä on seurata projektin etenemistä projektille määritetyn aikatauluarvioiden puitteissa. Mikäli mittaaja huomaa aikataulun venyvän, tulee hänen ohjata ryhmää tarkastamaan arvioita uudestaan ja tekemään tarvittavia muutoksia joko aikatauluun tai iteraatioihin.

Valmentaja (Coach) on henkilö, jolla on hyvät tekniset taidot ja kokemusta XP-prosessin läpiviennistä. Valmentaja huolehtii, että kehitysryhmä soveltaa XP:n prosessia, arvoja ja toimintatapoja niin, että se sopii projektin toteutusympäristöön.

Konsultti on liiketoiminnallisen tai teknisen alueen erikoisasiantuntija, joka voi auttaa projektiryhmää erityiskysymyksissä. Konsultti on kuitenkin ulkopuolinen henkilö ja hänen roolinsa projektissa on lähinnä vierailevа.

Johtajaa (Big Boss) voidaan kutsua myös projektipäälliköksi. Hänen päävastuualueensa on tehdä päätökset sekä huolehtia kehitysryhmän tarpeista, seurata edistymistä, reagoida muutoksiin ja poistaa etenemisen tiellä olevia esteitä. Johtajalta vaaditaan rohkeutta, luottamusta sekä tiettyä vaativuutta ohjata XP:n kehitysryhmää tekemään sitä, mitä he sanovat tekevänsä ja mitä heidän tuleekin tehdä. Johtaja laatii myös budjetin, hankkii tarvittavat resurssit ja työvälineet testausta varten sekä asianmukaiset tilat projektille. Beckin (2000, 147) omasta kokemuksesta johtajan ja

kehitysryhmän välisen kommunikaation tulee olla rehellistä ja johtajan tulee ymmärtää, ettei kehitysryhmä valita vaan yrittää selventää johtajalle tilanteita, jolloin asiat ovat poikkeamassa suunnitelmasta, jotta tällä olisi mahdollisimman paljon aikaa reagoida tilanteeseen.

4.5 Käytännön kokemuksia

Beckin (2000) mukaan XP soveltuu hyvin sellaisiin projekteihin, joissa vaatimukset eivät ole selvillä tai niiden uskotaan muuttuvan projektin aikana. Tällöin aikaa ei kuluteta esimerkiksi ylimääräiseen dokumentointiin ja suunnitteluun, sillä projektin etenemisen ennustaminen olisi joka tapauksessa vaikeaa. Lisäksi XP:n yksi perusominaisuuksista, nopea muutoksiin reagoiminen, tekee XP:stä hyvän menetelmän juuri tällaisiin projekteihin. Hyviä kohteita ovat projektit, joissa sovellusalue tai käytettävä tekniikka on uutta projektihenkilöstölle. Tiivis asiakasyhteystyö auttaa ymmärtämään aihetta puolin ja toisin, ja virheet huomataan nopeasti lyhyiden iteraatioiden kautta. Samalla tapaa myös uusi teknologia voi aiheuttaa ongelmia projektissa ja suunnitelmia voidaan joutua muuttamaan. Projektit, joiden tavoitteena on tuottaa nopeasti toimiva tuote asiakkaalle, soveltuvat myös hyvin XP:llä toteutettaviksi. Tässä luvussa esitellään muutamia XP:llä toteutettuja projekteja ja niistä saatuja tuloksia.

4.5.1 Case: Semper Corboration

Akbar Saeed ja Peter Tingling ovat tutkimuksessaan (Jacko 2007, 242–251) selvittäneet XP:n soveltuvuutta Semper Corporationin (myöh. Semper) käyttämässä projektissa. Tutkimuksen tiedot kerättiin elokuun 2005 ja joulukuun 2006 välisenä aikana, jolloin työntekijöitä haastateltiin säännöllisesti, tarkkailtiin kehittäjiä ja ohjelmoijia vähintään joka viikko joko paikan päällä tai kuvaamalla työskentelyä sekä tutustuttiin projektin tietoihin, dokumentteihin, virhelistoihin ja tuotteen eri versioihin. XP-menetelmän mukaiset käytännöt ja se, kuinka Semper pystyi projektissaan ne omaksumaan ja hyödyntämään, on kuvattu alla.

TAULUKKO 1. Semper Corporationin omaksumat XP:n käytännöt (Jacko 2007, 243)

eXtreme Programming Principle	Adoption Level*	Temporal Effects	Summary
40-Hour Work Week	Full	N	Developers worked flexible but regular workdays.
Coding Standards	Low to Partial	Y	Standards were initially avoided but later implemented.
Collective Code Ownership	Partial	Y	Code was officially shared but developers exhibited possessiveness.
Continuous Integration	Full	N	Code was rarely broken and was continually linked and compiled.
Continuous Testing	Partial to Full	Y	Testing was continuous but advance scripts were not created. Black box testing was phased.
On-Site Customer	Full	N	The CEO and Analytic Director acted as customers.
Pair Programming	Low	N	Programmers were independent except when difficulties or interdependencies existed.
Planning Game	Full	N	Value engineering balanced features against time and budget.
Refactoring	Full	Y	Modules were constantly improved. Periodic bursts of and dramatic improvement occurred.
Simple Design	Full	N	Working software was favored.
Small Releases	Full	N	Frequent (weekly) build cycles
System Metaphor	Full	N	Communication was simple and informal but unambiguous.

*Adoption is considered Complete, Partial or full

40 tunnin työviikko. Semperin yrityspolitiikkaan kuului liukuva työaika ja 40 tunnin työviikkoa noudatettiin, poikkeuksena muutama tapaus, jolloin tuntimäärä ylittyi asiakastestauksen tai ongelmien ratkaisun takia. Ylitöistä ei annettu hyvityksiä rahana tai vapaina. (Jacko 2007, 244.)

Koodistandardit (koodauskonventiot). Koodistandardeja välteltiin alusta alkaen ja keskityttiin luovuuteen, kunnes perustason koodi oli kehitetty. Esimerkiksi yksi ohjelmoija loi koodiin muuttujia sitä mukaa, kuin niitä tarvitsi, eikä määritellyt niitä heti

koodin alkuun standardin mukaisesti. Johto noteerasi pyynnöt standardien luomisesta vasta, kun ohjelmasta alkoi rakentua monimutkaisempi ja huomattiin, että ohjelmoijat jatkuvasti kirjoittivat muuttujia uudelleen refaktoroinnin tai moduulin vaihdon aikana. (Jacko 2007, 245.)

Koodin yhteisomistajuus. Muutamaa poikkeusta lukuun ottamatta aktiivisimmat kehittäjät tekivät päätökset toteutettavien moduulien osalta hiljaisempien tyytyessä päätöksiin. Tämän seurauksena yksittäisten osa-alueiden toteutuksessa moduuleja kirjoitettiin uudelleen. Vaikka moduuleilla oli useita tekijöitä, tuli tilanteita, joissa kehittäjät olivat vastahakoisia liittämään muiden tekemää koodia ”oman koodinsa” sekaan. (Jacko 2007, 246.)

Jatkuva integraatio. Ohjelmointikieli oli VB.NET ja ohjelmointikehys .NET. Moduuleja testattiin ja niitä lisättiin sitä mukaa ohjelmaan useita kertoja päivässä. Seurannan, joka kesti 16 kuukautta, aikana rakennettiin yli 35 valmista versiota lopputuotteesta ja integraatioita tehtiin 225. Toimivasta koodista huolimatta vastaan tuli muutamia tapauksia, joissa arkkitehtuuriin jouduttiin tekemään muutoksia ja samalla koodia jouduttiin kirjoittamaan uudelleen jopa kaksi viikkoa. (Jacko 2007, 246.)

Testilähtöinen kehitys. Testejä ei kirjoitettu ennen koodauksen aloittamista, toisin kuin XP suosittelee. Toiminnallisuus- ja yhteensopivuustestit käyttivät tiettyjä standardeja ja ad-hoc-testiskriptejä. Integraatiotestaus suoritettiin viikoittain johdon ja ulkopuolisten käyttäjien puolesta. Black box -testaus toteutettiin loppukäyttäjien ja esimerkkitestien avulla. Semper käytti ohjelmaa virheiden jäljitykseen useimmissa suuremmissa ongelmissa, mutta yleensä ohjelmoijat korjasivat havaitut virheet välittömästi. (Jacko 2007, 246.)

Asiakasyhteistyö. Koska Semper oli alkuvaiheessa oleva premarket-yritys, heillä ei ollut tavanomaisia asiakkaita. Tämän vuoksi tuotteen vaatimukset saatiin muun muassa toimitusjohtajalta. Toimitusjohtaja oli yleistason ohjelmoija, joten hän ymmärsi teknisiä ongelmia, mutta hänellä ei ollut tietämystä nykyisten järjestelmien rakentamisesta eikä hän osallistunut niitä koskeviin yksityiskohtiin. Kehitystiimi työskenteli

aktiivisesti hallinnosta vastaavien henkilöiden kanssa ja esitteli aikaansaannoksiaan saadakseen nopeaa palautetta. Kommunikointi oli pääosin suullista, mutta koska toimistot olivat fyysisesti toisistaan erillään, myös sähköpostia ja pikaviestimiä käytettiin. (Jacko 2007, 246–247.)

Pariohjelmointi. Pariohjelmointia ei käytetty missään vaiheessa ja moduulit kirjoitti aina yksi henkilö, joskin monimutkaiset ja hankalat ongelmat jaettiin koko ryhmän kesken. Vaikka johto keskusteli pariohjelmoinnin käytöstä yhtenä vaihtoehtona jo silloin, kun työntekijöitä palkattiin, sitä ei toteutettu. Työntekijät oli palkattu suoraan yliopistosta, joten heillä oli samanlainen tieto- ja taitoperusta. Tämä johti siihen, että he toisinaan kilpailivat siitä, kuka pystyy kirjoittamaan koodia tehokkaimmin. Pariohjelmointia ei voitu toteuttaa työntekijöiden erilaisten aikataulujen, henkilöstön vaihtuvuuden ja erilaisten persoonien vuoksi. Kaksi työntekijää esimerkiksi halusi kuunnella IPodejaan ja eristäytyä muusta ryhmästä. Koska koodi ei ollut työntekijöiden yhteisomistajuudessa, oli usein tilanteita, joissa työntekijän esimerkiksi vaihdettua työpaikkaa tämän tekemää koodia saatettiin kommentoida ”omituiseksi” tai ”huonosti kirjoitetuksi”. (Jacko 2007, 247.)

Suunnittelupalaverit. Tutkimuksen aikana suunnittelupalavereista tuli rutiinia ja niihin liittyviä käytäntöjä sovellettiin säännöllisesti. Suunnittelupalaverien pohjalta vaatimukset jaettiin seuraaviin ryhmiin: rakenteelliset vaatimukset, erottuvat ominaisuudet, käyttäjän ja tietokoneen väliseen vuorovaikutukseen liittyvät sekä kosmeettiset muutokset. Rakenteelliset vaatimukset olivat tärkeimpiä ydinominaisuuksia. Erottuvia ominaisuuksia olivat vaatimukset, jotka tekivät tuotteesta kilpailukykyisen ja erottivat sen muista vastaavista tuotteista. Näitä ominaisuuksia ryhmiteltiin vielä sen mukaan, olivatko ne pakollisia, hyviä lisäominaisuuksia vai ominaisuuksia, jotka tehtäisiin vain, jos siihen jäisi ylimääräistä aikaa. Käyttäjän ja tietokoneen väliseen vuorovaikutukseen liittyvät vaatimukset koskivat käytettävyyttä, ja sen toteuttamiseen palkattiin suunnittelijoita sen jälkeen, kun tuotteen ensimmäinen versio oli julkaistu. Kosmeettisia muutoksia, kuten tekstin fontin, värin, sijainnin tai tasauksen muuttamisia, tehtiin useita. Muutokset jatkuivat läpi projektin ja ne olivat kehittäjien mielestä pitkäveteisiä, mutta helppoja tehdä. (Jacko 2007, 247–248.)

Koodin refaktorointi. Koodi keskittyi toiminnallisuuteen ja sitä kehitettiin ja parannettiin jatkuvasti. Tuotteen ensimmäinen versio julkaistiin kahden viikon kuluttua projektin aloittamisesta. Tuotteeseen tehtiin muutamia suuria muutoksia. Esimerkiksi täydellinen muutos järjestelmän ulkoasussa vaati kaikkien moduulien uudelleenkirjoittamista. Tämä vaihe kesti kaksi kuukautta, jonka aikana koodia kirjoitettiin uudelleen yli 6000 riviä. (Jacko 2007, 248.)

Yksinkertainen suunnittelu ja julkaisut. Kehitystyössä korostettiin yksinkertaisen suunnittelun periaatteita ja toimitusjohtajan sanoin yritettiin välttää ”itsensä maalaamista nurkkaan”. Kehitystyön aikana ilmenneitä ongelmia kuvattiin kirjainyhdistelmillä BR ja AR. BR tarkoitti ongelmaa, joka vaikutti asiakkaaseen ja joka tuli korjata ennen julkaisua. AR-merkinnällä tarkoitettiin ongelmaa, jonka korjaaminen voitiin jättää julkaisun hoidettavaksi julkaisun jälkeen. Tuotteen uusin versio julkaistiin pääsääntöisesti joka toinen viikko. Vaikka kehittäjät muutamaan otteeseen ilmaisivat mielipiteensä siitä, että lyhyet iteraatiot eivät olleet paras käytäntö järjestelmällisen ja laadukkaan ohjelmiston tuottamiseen, johtoryhmä pysyi päätöksessään pitää iteraatiot lyhyinä. Vuoden aikana kehitysryhmä tuotti noin 35 valmista versiota tuotteesta. (Jacko 2007, 249–250.)

Metafora. Metafora tuotteelle oli ”nykyinen digitaalikamera”, jossa monimutkaiset toiminnot on kätketty yksinkertaisen ulkoasun sisään. Kaiken kaikkiaan kommunikaatio ryhmän sisällä oli yksinkertaista ja selkää johtuen siitä, että melkein kaikki työntekijät osasivat systeemisuunnittelua tai ohjelmointia. (Jacko 2007, 249–250.)

Yhteenveto

Alun perin Semper otti käyttöönsä vain kahdeksan käytäntöä. Kolme jäljelle jäänyttä (koodistandardit, jaettu koodi ja jatkuva testaus) tulivat kuitenkin tarkoituksenmukaisiksi ja niitä hyödynnettiin projektin edetessä. Pariohjelmointia ei edes kokeiltu koko projektin aikana. Aluksi näytti ehkä siltä, että Semperin olisi pitänyt olla huolellisempi ketterien menetelmien käyttöönotossa heti alusta alkaen. Saeed ja Tingling (Jacko 2007, 250) ovat kuitenkin sitä mieltä, että koodistandardien ja jaetun koodin käyttäminen sekä jatkuva testaus vaativat tietyn kypsyysasteen projektiryhmältä,

jotta niitä pystyttäisiin soveltamaan. Pariohjelmoinnin toteutukseen olisi ollut hyvät mahdollisuudet, sillä kehittäjillä olivat yhdenvertaiset tiedot ja taidot. Yhteenvetona voidaan todeta, että ketterien menetelmien osittainen käyttöönotto Semperin tapauksessa vahvistaa käsitystä siitä, että kaksi kolmesta suuresta yrityksestä hyödyntää jossain muodossa myös joitain ketteriä menetelmiä, joita on sitten yhdistetty perinteisempiin menetelmiin hyötyjen saamiseksi.

4.5.2 Case: ThoughtWorks Inc.

ThoughtWorks Inc. on yhdysvaltainen järjestelmäintegraatioihin ja konsultointiin keskittyvä yritys, joka työllistää noin 300 henkilöä. Sen erityisalaa ovat uuden tekniikan harjalla ratsastavien sovellusten tekeminen liiketoimintakäyttöön. ThoughtWorks työskentelee myös monimutkaisten sovellusten kanssa, jotka vaativat useammin erilaisen komponentin yhteen liittämistä ja kehittämällä komponentteja vastaamaan jatkuvasti kehittyvää liiketoimintaa. (Cara & Fowler 2000, 13.) Tässä luvussa on esitelty yksi ThoughtWorksin projekti, jossa he ottivat XP:n ensimmäistä kertaa käyttöönsä.

Alkutilanne

Aluksi Thoughtworks rakensi laajaa Java-sovellusta perinteisiä prosessimenetelmiä käyttäen. Sovelluksen kokoa kuvaa hyvin se, että ThoughtWorks käytti puolitoista vuotta pelkästään vaatimusten keräämiseen ja prototyyppien luomiseen, kunnes pääsi aloittamaan varsinaisen kehitystyön. Tämän aikana he kohtasivat useita ongelmia, jotka ovat tyypillisiä projekteissa, joissa vaatimukset eivät ole selkeitä tai ne muuttuvat projektin aikana. Projektin hallinta osoittautui hankalaksi, sillä se oli hallinnollisesti ja toteutukseen liittyvien vaatimusten osalta todella laaja. Jossain vaiheessa syntyi idea XP:n hyödyntämisestä projektissa; sen ominaisuudet tuntuivat istuvan perinteisiä menetelmiä paremmin projektin läpivientiä vaativiin prosesseihin. (Cara & Fowler 2000, 13.)

XP:n käyttöönotto

XP:n käyttöönoton alkuvaiheessa oli selvää, että ensimmäisen asiakasta hyödyntävän

version tuottamiseen tulee kulumaan paljon aikaa, sillä täysin uuden menetelmän omaksuminen ei tapahdu hetkessä. Koska oli selvää, ettei XP:tä pystyttäisi toteuttamaan täysin sen sääntöjen mukaisesti, päädyttiin siihen, että menetelmiä tultaisiin muuttamaan tarvittaessa. Näin ne vastaavat yrityksen ja projektin tarpeita parhaalla mahdollisella tavalla. (Cara & Fowler 2000, 13.)

Projektiryhmä

Projektiryhmään kuului 50 henkilöä, joista puolet oli ohjelmistokehittäjiä. Projektin kehittäjistä koostuva henkilöstö jaettiin kahteen tiimiin, joista toisen vastuulle määrättiin puhtaasti toiminnallisuus eli koodaus. Toinen tiimi, SWAT-tiimi, oli vastuussa virheiden korjaamisesta, testauksen automatisoinnista ja muista kehittämistehtävistä. Ryhmäjako perustui pyrkimykseen säilyttää yhtenäisyys suunnittelussa ja pitää työntekijät omissa rooleissaan. Ryhmien henkilöt vaihtuivat niin, että jokainen kuului jossain projektin vaiheessa myös toiseen tiimiin. Kaksi iteraatiomanageria määrättiin huolehtimaan heille vuoronperään annettavien iteraatioiden toiminnallisuudesta. Iteraatiomanagereille annettiinkin iteraatioiden järjestystä kuvaavat nimet ”pariton” (Odd) ja ”parillinen” (Even). Projektin managereiksi valittiin kaksi henkilöä, joiden tehtävänä oli huolehtia asiakasyhteistyöstä. Muita henkilöitä projektiryhmässä olivat muun muassa laaduntarkistustiimi sekä tekniset asiantuntijat. (Cara & Fowler 2000, 13–15.)

Asiakas

Asiakas oli 12 henkilön tiimi, johon kuului tietotekniikan ammattilaisia ja henkilöitä, jotka suorittavat tuotteen beta-testauksen. Asiakas oli mukana iteraatioiden suunnittelussa ja pääsi vaikuttamaan toteuttaviin vaatimuksiin ja niiden toteuttamisjärjestykseen, kuten XP:n sääntöihin kuuluu. Asiakas sai luonnostella seuraavia toiminnallisuuksia, joita ei vielä ollut lisätty julkaisusuunnitelmaan, ottaa mukaan toiminnallisuuksia, jotka oli siirretty toteutettavaksi myöhemmin, jättää toiminnallisuuksia kokonaan pois tai priorisoida käyttäjätarinoita uudelleen. Kaikki tämä sallittiin, kunhan se ei ajallisesti tuottanut ongelmia. Koska kyseessä oli mittava projekti, seuraavan iteraation toiminnallisuuksien valintaan kului yleensä noin viikko. (Cara & Fowler 2000, 15.)

Iteraation prosessi

Cara ja Fowler (2000) kuvaavat käyttämäänsä XP:tä esittelemällä yhden iteraation, iteraation numero 6. Iteraatio 6:n kesto oli yksi kuukausi, mutta todellisuudessa osa sen toiminnoista tapahtui jo sitä ennen ja myös sen jälkeen. Iteraatio aloitettiin suunnittelupalaverilla heti sen jälkeen, kun iteraatio 5:n tuotantovaihe oli aloitettu. Suunnittelupalaverissa kehitystiimi päätti, mitä toimintoja iteraatio 5 tulee tuottamaan, jolloin se tiesi, mitkä käyttäjätarinat ovat toteutettavissa iteraatiossa 6. Myös asiakas oli tässä vaiheessa mukana. Kuten on todettu, päällekkäin toimivien iteraatioiden hallintaan käytettiin kahta iteraatiomanageria. Iteraatioiden 5–7 suunnitellut prosessit on nähtävissä alla olevasta taulukosta. Taulukosta nähdään, että seuraavan iteraation suunnittelu aloitettiin jo edellisessä iteraatiossa, jossa myös toimitettiin sitä edeltävä iteraatio. (Cara & Fowler 2000, 14–15.)

TAULUKKO 2. Iteraatioiden 5–7 prosessit (Cara & Fowler 2000, 14)

Iteration 5 Planning Meeting	Get List of Story Cards for Iteration 6	Plan Resources for Iteration 6	Deliver Iteration 4	Close Iteration 5	Iteration 6 Planning Meeting	Get List of Story Cards for Iteration 7	Plan Resources for Iteration 7	Deliver Iteration 5	Close Iteration 6	Iteration 7 Planning Meeting	Get List of Story Cards for Iteration 8	Plan Resources for Iteration 8	Deliver Iteration 6	Close Iteration 7
	Manage Iteration 5					Manage Iteration 6					Manage Iteration 7			
	Prepare for Delivery of Iteration 4	Work with Analysts to Articulate Iteration 6 Functionality		Prepare for Iteration 6 Meeting		Prepare for Delivery of Iteration 5	Work with Analysts to Articulate Iteration 7 Functionality		Prepare Iteration 7 Meeting		Prepare for Delivery of Iteration 6	Work with Analysts to Articulate Iteration 8 Functionality		Prepare Iteration 8 Meeting
	Week 1	Week 2	Week 3	Week 4		Week 1	Week 2	Week 3	Week 4		Week 1	Week 2	Week 3	Week 4
	Iteration 5					Iteration 6					Iteration 7			

Suunnittelupalaverit

Yhden päivän kestävään suunnittelupalaveriin osallistui aina koko projektiryhmä, 50 henkeä. Ryhmä kokoontui isoon huoneeseen, jonka seinille oli kiinnitetty julisteiden kokoisia käyttäjätarinoita. Käyttäjätarinoita lähdettiin purkamaan seuraavana olevan käyttäjätarinan kautta. Tällöin keskusteltiin käyttäjätarinan vaatimasta skenaariosta ja toiminnallisuudesta. Kehittäjien rooli oli tehdä lista toteuttavista osista. Listalla olevat osat kirjattiin erillisille lapuille, jotka kiinnitettiin kyseiseen käyttäjätarinaan (ks. alla oleva taulukko). Kun toiminnallisuudet oli valittu, valittiin niiden tekijät. Perussääntönä pidettiin sitä, että jokaisen kehittäjän tulisi valita vain muutama tehtävä, eikä yhdelläkään tehtävällä tulisi olla liian monta tekijää. Suunnittelupalaverin, joka

saattoi olla hyvinkin raskas työntekijöille, jälkeen kaikilla oli tiedossa seuraavan kuukauden suunnitelma ja tavoite, joka säilytettiin julisteisiin kirjoitettuna seinällä koko iteraatioon ajan. (Cara & Fowler 2000, 16.)

TAULUKKO 3. Esimerkki käyttäjätarinasta ja sen tehtävälistasta (Cara & Fowler 2000, 16)

Story Card

Dispose asset		
Create A/R record for sales price with or without tax calculation. Create journal entries for A/R record. Create expenses or A/R records to handle disposal costs & appropriate journal entries. Change asset status to "disposed." Create journal entries for relieving (fixed asset and accumulated) inventory and calculated gain or loss. Set stop dates on book and tax depreciation. Create transaction history record. Asset must be active-in inventory.	2 Weeks	Analyst: Terri Developer: Biren QA: Shelly

Task List

Write dispose asset transactor	1 day	Biren
Make journal entries	2 days	Biren
GUI - Create asset disposal screen	3 days	Caleb
Calculate gain/loss	2 days	Biren
Create pessimistic session bean	1 day	Biren
Generate disposal charge	1 day	Biren
Update statuses	1 day	Biren
CFT	2 days	Caleb

Iteraation toteutus

Toiminnallisuudesta vastaavan tiimin vastuulla oli huolehtia iteraation toiminnallisista vaatimuksista, ja henkilötasolla jokainen vastasi niistä tehtävistä, joihin suunnittelupalaverissa oli ilmoittautunut. Tiimiläisten, eli kehittäjien, ainoa tehtävä iteraation aikana oli tuottaa vaadittu toiminnallisuus. He työskentelivät tiiviissä yhteistyössä asiantuntijoiden kanssa, jotka olivat jatkuvasti tiimin käytettävissä. Joka toinen päivä kehittäjät kokoontuivat nopeaan palaveriin, jossa keskusteltiin menossa olevista teh-

tävistä ja ratkaistiin ongelmia ryhmässä. Vaikka yksittäinen kehittäjä vastasikin sen tehtävän valmistumisesta, johon hän ilmoittautui, toteutettiin koodaus pariohjelmointina. Koodin osia varten kirjoitettiin yksikkötestit, joiden läpäisemisen jälkeen koodi voitiin liittää varsinaiseen ohjelmiston versioon. Toinen tiimi, SWAT-tiimi, puolestaan oli vastuussa kaikista niistä toiminnallisista osista, jotka eivät liittyneet uusimpaan meneillä olevaan iteraatioon. Heidän tehtävänsä oli huolehtia siitä, että kehittäjätiimi pystyy keskittymään ainoastaan uusien toiminnallisuuksien tuottamiseen. Käytännössä tämä tarkoitti esimerkiksi edellisten iteraatioiden virheiden korjaamista, automatisoidun testauksen hoitamista sekä suorituskykyyn liittyvien toimintojen hoitamista. (Cara & Fowler 2000, 17–18.)

Julkaisut

Iteraation lopussa laadusta vastaava tiimi teki regressiotestit varmistaakseen toiminnallisuuden laadun. Heidän vastuullaan oli myös hyväksyä käyttäjätarinat toteutetuiksi. Käyttäjätarinoiden hyväksymisessä oli mukana laadun tarkkailijoiden lisäksi asiantuntija, kehitystiimin jäsen sekä SWAT-tiimi varmistamassa, että käyttäjätarinoita varten kirjoitetut testit vastasivat toiminnallisuutta, läpäisivät testit ja että toteutettu uusi toiminnallisuus voidaan välittää asiakkaalle. Laaduntarkkailutiimi loi yhdessä projektipäällikön ja asiantuntijoiden kanssa toiminnallisuuksia kuvaavan tarpeellisen dokumentoinnin, joka luovutettiin asiakkaalle version yhteydessä. Dokumentointi kattoi käyttäjätarinat, iteraation testit ja koko projektin kehityksen siihen mennessä. Laaduntarkkailutiimi myös esitteli toiminnallisuuden ja koulutti tarvittaessa käyttäjiä. (Cara & Fowler 2000, 18.)

Ongelmat

Caran ja Fowlerin (2000, 20–21) mukaan projektin aikana ilmeni ongelmia liittyen muun muassa aikataulutukseen, vaatimusten määritykseen ja suuren projektiryhmän hallinnointiin. Aikataulutuksen suhteen ongelmallista oli suunnitella jokaiselle tiimin jäsenelle järkevä aikataulu. Asiakkaalla oli myös hankaluuksia vaatimusten määrittämisessä ja niiden selkeässä esiintuonnissa. Myöskään projektin työntekijöillä ei ollut antaa vastausta siihen, miten kauan niin sanotun hyvän vaatimuksen laatimiseen menee, vaan asioita opittiin puolin ja toisin. Päivän kestävät suunnittelupalaverit

koettiin usein kauheina ja työläinä, vaikka ne olivatkin varsin tuottavia. Projektin aikana huomattiin, että palavereiden parantamiseksi käyttäjätarinoiden tekemiseen kannatti kiinnittää erityistä huomiota. Suuren projektiryhmän ja laajan sovelluksen muodostamaa kokonaiskuvaa siitä, mitä toiminnallisuuksia on tehty, miten ne toimivat nyt ja kuinka kaiken pitää toimia lopulta yhdessä, oli välillä vaikea hahmottaa. Myös kommunikaation kanssa tuli ajoittain ongelmia lähinnä kulttuurin ja projektiryhmän suuren koon vuoksi.

Lopputulokset ja yhteenveto

Cara ja Fowler (2000, 18–19) toteavat yhteenvetona, että XP:n käyttäminen projektissa oli onnistunut. Tästä esimerkkinä he mainitsevat, että arviot siitä, kuinka paljon toiminnallisuuksia saadaan toteutettua kunkin iteraation aikana, pitivät todella hyvin paikkaansa. Jokaisena kuukautena he saivat aikaan näkyvää tulosta ja projekti eteni pienissä jaksoissa, mutta varmasti. Asiakkaan, työntekijöiden ja sponsoreiden välinen kommunikaatio parantui huomattavasti. Suunnittelupalavereita pidettiin toimivina käytäntöinä, sillä kaikki osallistuivat siihen ja olivat näin ollen perillä projektin edistymisestä ja sen seuraavista päämääristä. Heidän mukaansa työntekijöiden yhteishenki parantui huomattavasti ja he pystyivät parantamaan toimintaansa projektin aikana.

Projektin aikana esiintyviin ongelmiin löytyi usein ratkaisu tai ne pystyttiin muutoin hyväksymään niin, että ne eivät vaikuttaneet projektin etenemiseen. XP:n avulla ThoughtWorksin onnistui löytää niitä prosessinhallintamenetelmiä ja käytäntöjä, jotka sopivat heidän projekteihinsa. Tärkeää oli omaksua ajatus siitä, että XP:tä voidaan käyttää eräänlaisena aloituspisteenä, josta käytettävää prosessia voidaan lähteä muokkaamaan niin, että se soveltuu käytettävään projektiin ja sen työntekijöille. (Cara & Fowler 2000, 18-19.)

5 TIETOKONEPAKETTILASKURIN TOTEUTUS

Tässä luvussa kuvataan tietokonepakettilaskurin toteutus. Ensimmäisessä alaluvussa esitellään toteutusympäristö, eli käytetyt tekniikat ja ohjelmistot. Tietokonepakettilaskurin käytännön toteutus on kuvattu luvussa 5.2–5.5.

5.1 Toteutusympäristö

Sovelluksen toteutukseen valittiin .NET Frameworkin laajennus ASP.NET, koska se on joustava ja tarjoaa paljon hyviä valmiita komponentteja ja kirjastoja. Sovelluksen toteutuskieleksi valittiin VB.NET. Visual Studion kautta sovelluksen kehitys pystyttiin pitkälti tekemään graafisesti. Alla on kuvattu käytetyt tekniikat ja ohjelmisto lyhyesti.

Microsoft .NET Framework

.NET Framework on Microsoftin kehittämä ohjelmointikehys, joka sisältää useita valmiita komponentteja, ratkaisuja ja kirjastoja ohjelmointia varten. Sydänheimon (2009, 20–21) mukaan .NET-arkkitehtuuri koostuu kahdesta pääosasta: Common Language Runtime (CLR) ja .NET Framework Class Library (.NET CL). CLR tarjoaa palvelut ajettavien ohjelmien taustalla ja on näin perustana .NET-ympäristölle. CLR huolehtii muun muassa käännetyn ohjelmakoodin ajamisesta, ohjelmasäikeiden hallinnasta, ajonaikaisesta turvallisuudesta ja käyttövarmuudesta. .NET CL on kokoelma yleiskäyttöisiä luokkia. .NET Framework tukee useita ohjelmakieliä, joista käytetyimmät ovat C# ja Visual Basic .NET (VB.NET). .NET Frameworkin uusin versio on 3.5 Service Pack 1. (Microsoft 2010). Uusi versio, .NET Framework 4.0 oli projektin aikana vielä kehitysvaiheessa.

Microsoft ASP.NET

ASP.NET on .NET-ohjelmointikehyksen laajennus dynaamisten web-sovellusten luomiseen. AJAX-tekniikka (Asynchronous JavaScript and XML) tuo ASP.NETtiin lisää mahdollisuuksia persoonallisten, interaktiivisten ja tehokkaiden sovellusten rakentamiseen. ASP.NETin viimeisin vakaa julkaistu versio on 3.5 Service Pack 1. (Microsoft 2009.) ASP.NET on selainriippumaton ohjelmointimalli, joten sillä tehtyjä sovelluksia

voidaan suorittaa kaikilla yleisesti käytössä olevilla selaimilla. Tämä helpottaa kehittäjien työtä, sillä heidän ei tarvitse käyttää aikaa selainkohtaisen koodin kirjoittamiseen. (Sydänheimo 2009, 20.)

Microsoft Visual Studio 2008 Professional

Visual Studio on Microsoftin ohjelmistokehitysympäristö, jolla voidaan luoda muun muassa Windows-, web- ja mobiilisovelluksia. Ohjelmisto tukee useita ohjelmointikieliä, kuten C#:a ja VB.NETtiä. Graafisten käyttöliittymien luominen Visual Studiolla on nopeaa ja helppoa. Visual Studion viimeisin versio on Visual Studio 2008. Uusi versio, Visual Studio 2010, oli projektin aikana vielä kehitysvaiheessa.

5.2 Tutkimusvaihe

Beckin (2000, 133) mukaan tutkimusvaiheen tulisi kestää muutamia viikkoja silloin, kun projektiryhmä osaa käytettävän teknologian. Mikäli ryhmällä ei ole tarvittavia teknillisiä taitoja, tutkimusvaiheeseen olisi hyvä käyttää muutama kuukausi. Jos muutama kuukausikaan ei riittäisi, paras keino lähestyä aihetta olisi toteuttaa ensin pienempi projekti, jonka projektiryhmä osaa toteuttaa helposti ja siirtyä vasta sen jälkeen haastavampiin projekteihin. Tämän opinnäytetyön tekeminen aloitettiin tammikuussa 2010. Aloitettaessa käytettävät teknologiat (ASP.NET, VB.NET) eivät olleet tekijälle juurikaan entuudestaan tuttuja. Alkuvaiheessa asennettiin tarvittavia ohjelmistoja sekä tutustuttiin ASP.NET-teknologiaan ja VB.NET-ohjelmointikieleen tekemällä erilaisia harjoituksia. Kun käytettävä tekniikka oli tullut jollain tapaa tutuksi, alettiin muotoilla varsinaisen sovelluksen prototyyppiä. Pian tämän jälkeen pidettiin asiakkaan kanssa ensimmäinen suunnittelupalaveri. Kokonaisuudessaan tutkimusvaihe kesti noin kaksi kuukautta.

5.3 Suunnitteluvaihe

Yleinen aikataulu

Ensimmäisessä suunnittelupalaverissa käytiin läpi tulevaa prosessia ja asiakkaan roolia sen aikana. XP-menetelmä ei ollut asiakkaalle entuudestaan tuttu, joten siihen

tutustuttiin yhdessä asiakkaan kanssa ja sovittiin, että menetelmiä muokataan tarpeen vaatiessa niin, että ne sopivat tähän projektiin. Aikataulun suhteen asiakas ei määrännyt tarkkaa takarajaa, joten aikataulun laatimisessa päätettiin edetä niin, että sitä tarkennetaan jokaisen iteraation jälkeen. Opinnäytetyön palauttamisen puitteissa asiakkaan kanssa kuitenkin sovittiin, että sovelluksen tulee olla valmis viimeistään huhtikuun 2010 lopussa.

Käyttäjätarinat

Sovelluksen toiminnallisuutta koskevat vaatimukset olivat pääpiirteittäin jo projektin alkuvaiheessa tekijälle hyvin selvillä, olihan sovellusta rakennettukin jo jonkin aikaa ennen käyttäjätarinoiden kirjoittamista. Tästä johtuen asiakkaan kanssa käyttäjätarinoita mietittäessä pyrittiin paremminkin löytämään yhteinen ajatus siitä, millaisia järjestelmän toiminnot käytännössä tulisivat olemaan. Beckin (2000, 133) mukaan ensimmäiset käyttäjätarinat eivät useimmiten ole sellaisia, joita todellisuudessa tarvitaan. Kehitysryhmän onkin tärkeää antaa asiakkaalle paljon nopeaa palautetta, jotta käyttäjätarinoita voidaan täydentää. Joidenkin käyttäjätarinoiden kohdalla ohjelmoija voi joutua hieman kokeilemaan, mitä asiakas kyseisellä vaatimuksella tarkoittikaan. Beckin havainnot toistuivat myös tämän projektin aikana. Käyttäjätarinoiden merkitystä ja sisältöä käytiin asiakkaan kanssa yhdessä läpi, ja tämän pohjalta asiakas kirjoitti käyttäjätarinat. Yhteensä asiakkaalta saatiin 15 käyttäjätarinaa, joista muutama esimerkki alla, niin kuin asiakkaan edustaja (Aro 2010) on ne kirjoittanut.

Ulkoasuun liittyvä käyttäjätarina:

> *Välilehdillä on halpa perus työasema, keskihintainen ja kallis pelikone.*

Toiminnallisuuteen liittyvä käyttäjätarina:

> *Vaihtoehdot löytyvät vetovalikosta joka avautuu kun klikkaa esim. nuolta komponentin nimen jälkeen.*

Tilaukseen liittyvä käyttäjätarina:

> *Uudella sivulla olisi ”lähetä” painike josta painamalla tilaus lähtisi sähköpostitse myyjälle ja vahvistus tilauksesta asiakkaalle.*

Käyttäjätarinat olivat yksinkertaisia kuvauksia niistä toiminnoista, joita sovellukselta odotettiin. Käyttäjätarinoita ryhmiteltiin yhteen niiden samankaltaisuuden perusteella ja niitä myös muokattiin projektin edetessä. Kun muutama iteraatio oli valmis, huomattiin, ettei joitain käyttäjätarinoita tarvittu sellaisinaan, joten ne muutettiin sopivimmiksi. Yhtenä esimerkkinä tästä oli ominaisuus, joka näyttää tietokonepaketin kokonaishinnan. Alkuperäisen käyttäjätarinan mukaan kokonaishinta näytetään, kun käyttäjä painaa painiketta. Ominaisuus toteutettiin kuitenkin niin, että hinta päivittyy aina, kun käyttäjä vaihtaa tuotteen toiseen.

5.4 Iteraatiot

Beckin (2000, 133) mukaan ensimmäisen julkaisun aikataulun olisi hyvä olla kahdesta kuuteen kuukautta, sillä lyhyemmässä ajassa ei ehditä selvittää kaikkia mahdollisia ongelmia. Suunnittelupalaverissa asiakkaan kanssa päätettiin, että iteraatioiden kesto on yksi viikko niin, että iteraatio aloitetaan maanantaina ja se esitellään perjantaina aamupäivästä. Jokaisessa iteraation esittelyssä asiakkaalta tuli paljon palautetta ja kehitysehdotuksia. Esittelyssä ilmi tulleita muutosehdotuksia korjattiin jo perjantaina tai ne siirrettiin seuraaviin iteraatioihin. Käytännössä esittelytilanne loppui aina suunnittelupalaveriin, sillä esittelyn jälkeen asiakkaan kanssa käytiin koko projektin etenemistä läpi, päätettiin seuraavasta iteraatiosta ja suunniteltiin se. Koska sovelluksen tekeminen aloitettiin jo osittain tutkimusvaiheessa, ensimmäisen julkaisun aikataulu oli myös viikko. Otettaessa huomioon se aika, jonka tekijä käytti tarvittavien teknologioiden, harjoitusten ja tätä kautta sovelluksen pohjan rakentamiseen, voidaan todeta ensimmäisen iteraation ja siihen sisältyvän tutkimuksen kestäneen Beckin mainitsemat noin kaksi kuukautta.

Käyttäjätarinoiden niputtamisen yhteydessä ne siirrettiin Excel-taulukkoon, johon projektin aikataulua rakennettiin. Taulukkoa täydennettiin suunnittelupalavereiden pohjalta niin, että jokaisen palaverin jälkeen siitä kävi ilmi jo toteutetut vaatimukset sekä seuraavat iteraation lyhyt kuvaus. Taulukko alla havainnollistaa lopullista Excel-dokumenttia.

TAULUKKO 4. Projektin aikana toteutetut iteraatiot

	Nro	Vko	Pvm	Iteraatiot	Esittely	
MAALIS	1	9	1.-5.3.	Toiminnallisuus: Konepaketin valinta	PE	5.3.
	2	10	8.-12.3.	Toiminnallisuus: Komponentit	PE	12.2.
	3	11	15.-19.3.	Tilaus-sivu	PE	19.3.
	4	12	22.-26.3.	Toiminnallisuus: Tilauksen lähettäminen, validointi	PE	26.3.
HUHTI	5	13-14	29.3.-9.4.	Ulkoasu: Tekstit ja muu ulkoasu, viimeistely	PE	9.4.
	6	15	12.-16.4.	Toiminnallisuus: Tilaustietojen lähettäminen, sovelluksen luovutus	PE	16.4.

Ensimmäisen iteraation osalta päätettiin, että sovelluksen ulkoasun tulee olla lähellä lopullista ja että käyttäjä voi valita kolmesta eri tietokonepaketista. Tämä tarkoitti käytännössä sitä, että sovellukseen toteutettiin kolme eri sivua jokaiselle konepaketille ja jokaiselle sivulle luotiin tarvittavat osiot, joihin jatkossa tultaisiin lisäämään tarvittavat ominaisuudet ja toiminnallisuudet. Näitä osioita ovat komponentit, oheislaitteet, ohjelmistot ja lisäpalvelut. Jokaiselle sivulle toteutettiin myös alasvetovalikot sisältöineen. Toisen iteraation pääpaino oli yhdistää sovellus tietokantaan ja hakea komponentit sieltä tiettyjen ehtojen mukaisesti. Kolmannessa iteraatiossa tehtiin Tilaus-sivu, joka sisälsi yhteenvedon tilauksesta sekä lomakkeen käyttäjän yhteystietojen syöttöä varten. Neljännes iteraatio sisälsi tilauksen lähettämisen asiakkaan edustajan sähköpostiin sekä yhteystietolomakkeen validoinnin. Viides iteraatio oli poikkeuksellisesti kahden viikon mittainen. Tämän iteraation aikana asiakas täydensi sovelluksen tekstejä ja tekijä viimeisteli ulkoasua ja muita toiminnallisuuksia sekä refaktoroi koodia. Kuudenteen iteraatioon asiakas halusi lisätä vielä tilaustietojen lähettämisen myös käyttäjän sähköpostiin. Toiminnallisuus toteutettiin vaatimusten mukaisesti. Lopuksi sovellus luovutettiin asiakkaalle. Varsinainen käyttöönotto jäi asiakkaan vastuulle, sillä sovellusta ei voitu lisätä asiakkaan web-sivustolle, koska tarvittavat järjestelmät eivät olleet tässä vaiheessa vielä valmiita.

Lindbergiin (2003, 41) viitaten XP:n iteratiivista mallia toteutettiin niin, että iteraatioiden jälkeen asiakkaalle ei vielä annettu ohjelmistoa käyttöön, vaan tarkoituksena oli tarkistaa siihen asti tuotetut osat. Tällä tavalla ohjelmistosta saatiin säännöllisesti palautetta ja virheet sekä väärinymmärrykset tulivat ilmi. Päätös siitä, että sovellusta ei oteta käyttöön, ennen kuin sen on lopullisesti valmis, tuli asiakkaan puolelta ja oli tämänkaltaisessa tapauksessa täysin ymmärrettävä. Kuvakaappaukset lopullisesta sovelluksesta löytyvät liitteistä 1-2.

5.5 XP:n menetelmien hyödyntäminen ja omaksuminen

Jo projektin alussa oli selvää, ettei läheskään kaikkia XP:n menetelmiä voida käyttää projektin aikana. Tämä oli harmillista, mutta projektiin yritettiin mahdollisuuksien mukaan liittää niitä käytäntöjä, jotka siihen sopivat, tai muokata käytäntöjä tilanteen vaatimalla tavalla. Tässä luvussa kerrotaan yksittäisistä käytännöistä ja niiden hyödyntämisestä projektin aikana. Osa käytänteistä on tullut ilmi jo edellisissä luvuissa 5.2-5.4, joten niihin ei enää tässä palata.

Metafora

Virallista metaforaa sovellukselle ei annettu, mutta sana ”pakettilaskuri” kuvasi sovellusta hyvin. Ensimmäisen suunnittelupalaverin aikana asiakas muotoili lisäksi myös niin sanotun visiokortin, johon hän kuvasi sovelluksen alle 25 sanalla. Alla asiakkaan edustajan muotoilema visiokortti.

> *Järjestelmä jonka avulla asiakkaat pääsevät itse muokkaamaan myymiämme konepaketteja nettiselaimen kautta. Asiakkaalle näkyviä tietoja/valikoita päästään hallitsemaan käytössä olevan [nimi poistettu] ohjelmiston kautta.* (Aro 2010.)

40 tunnin työviikko

Projektin aikana 40 tunnin työviikkoa noudatettiin pääsääntöisesti. Toisinaan työpäivä saattoi olla lyhyempikin, mikäli vaatimusten mukainen iteraatio saatiin valmiiksi etuajassa. Koko projektin aikana, ja etenkin alussa tutkimusvaiheessa, oli päiviä, jolloin mitään näkyvää ei käytännössä saatu aikaan, vaan koko päivä kului ongelmien

ratkomiseen. Tämä oli tekijän kannalta välillä todella turhauttavaa. Eniten tällaisia päiviä tuli tutkimusvaiheessa, kun sovelluksen toteutuksen suhteen vaihdettiin toisenlaiseen tapaan. Tästä lisää seuraavassa kappaleessa.

Testilähtöinen kehitys

Sovelluksen teossa ei toteutettu XP:n suosimaa testilähtöistä kehitystä. Syynä tähän olivat tekijän puutteelliset tekniset taidot. Projektin alussa lähdettiin tutustumaan .NET-ohjelmointikehyksen ASP.NET WebForms-malliin ja VB.NET-ohjelmointikieleen, joilla sovellus oli tarkoitus toteuttaa. Hyvin pian huomattiin, että sovelluksen toteutus ASP.NET-tekniikalla osoittautui hyväksy ratkaisuksi, sillä se tarjoaa monipuolisia ja helppokäyttöisiä ratkaisuja tarvittavien ominaisuuksien toteuttamiseen. Kun projektiin alettiin ottaa enemmän XP:n käytäntöjä mukaan, huomattiin, että testilähtöinen kehitys vaatisi ASP.NET MVC -mallin käyttämistä, sillä WebForms-ohjelmointimallilla testejä ei voitu tehdä etukäteen. MVC-mallin idea oli tekijälle tuttu aikaisemmista opinnoista, mutta sen soveltaminen käytännössä ei. MVC-mallia lähdettiin kuitenkin rohkeasti opiskelemaan ja kokeilemaan, jolloin samalla huomattiin, että ohjelmointikieli olisi järkevää vaihtaa C#-kieleen, sillä suurin osa MVC:tä koskevasta ohjeistuksesta oli toteutettu C#-kielellä. Tämä palautti projektin käytännössä alkuvaiheeseen, jossa käytettävät tekniikat tuli opetella uudestaan. Seuraavassa on esitelty ASP.NETin WebForms-mallin ja MVC-mallin eroavaisuudet lyhyesti.

Hanselmanin (2010) mukaan ohjelmointimallin valintaan vaikuttaa se, mikä tuntuu kehittäjälle luonnollisemmalta, ja se, kumpi malli soveltuu paremmin sovelluksen rakenteeseen. WebForms on kontrolli- ja tapahtumapohjainen malli, joissa eri kontrolleihin voidaan sisällyttää HTML-kieltä, Java Scriptiä tai CSS-tyylittelyjä. WebForms sisältää myös kontrolleja, jotka mahdollistavat muun muassa tietokannasta haetun tiedon esittämisen valmiissa DataGrid-kontrolleissa tai kaavioissa. MVC-malli (Model View Controller) on tutumpi ohjelmistokehittäjille, jotka ovat tottuneet käyttämään perinteisempiä tekniikoita. MVC-mallin tarkoituksena on erottaa käyttöliittymä sovellusaluetidoista ja siinä HTML-kieltä voidaan hallita täydellisesti. MVC tukee myös yksikkötestausta, testilähtöistä kehitystä ja sitä myöten myös erittäin hyvin ketteriä ohjelmistokehitysmenetelmiä. MVC on WebFormsia joustavampi ja paremmin laa-

jennettavissa oleva malli. Malleja voidaan käyttää myös yhtä aikaa, jolloin osa sovelluksesta toimii WebForms-mallin kautta, osa MVC:n. Mallien yhteiskäyttö on tavanomaista ja sitä kautta voidaan saada molemmista ne hyödyt, jotka sovelluksen rakentamiseen tarvitaan.

Tutkimusvaiheesta suurin osa ajasta käytettiin MVC-mallin ja sitä kautta testilähtöisen kehityksen opiskeluun. Tekijän kokonaiskuva siitä, mitä tulisi tehdä, oli selkeä. Käytännössä se, kuinka kaikki toteutetaan, jäi kuitenkin epäselväksi tässä tapauksessa välttämättömien teknisten taitojen puuttuessa. XP:n käytäntöihin tutustuttaessa kävi ilmi, että testilähtöinen kehitys vaatii hyvää teknistä osaamista ohjelmointikielen osalta ja tekniikoiden hallintaa. Sovelluksen toteuttaminen WebForms-mallilla oli yksinkertaisempaa, eikä se vaarantanut projektin etenemistä. Tämän vuoksi MVC-malli ja sitä myöten myös testilähtöinen kehitys päätettiin jättää projektin ulkopuolelle. Asiakkaalle teknisellä toteutuksella ei ollut merkitystä, joten päätöksen teki yksin tekijä.

Sovelluksen normaali testaus pyrittiin toteuttamaan kehitystyön rinnalla aina, kun virheitä havaittiin ja ennen iteraation esittelyä asiakkaalle. Käyttäjätestauksesta vastasi pääasiassa asiakas. Kolmannen iteraation (ks. taulukko sivulla 46) päätyttyä asiakkaalle luovutettiin kopio senhetkisestä sovelluksesta, jota asiakas tämän jälkeen testasi itsenäisesti ja ilmoitti virheistä ja muista huomautettavista asioista ennen seuraavan iteraation loppumista. Käyttäjätestauksella pyrittiin todentamaan asetettujen vaatimusten täyttyminen ennen sovelluksen varsinaista julkaisua.

Koodistandardit ja koodin yhteisomistajuus

Projektin aikana sovelluksen koodissa pyrittiin pitämään tietty samanlaisuus, jotta rakenne pysyisi selkeänä. Koodia kommentoitiin säännöllisesti luonnollisena osana sen tuottamista. Koska tekijöitä oli vain yksi, ei koodia jaettu kenenkään toisen kanssa, joten yhteisomistajuuteen ei voida tässä tapauksessa ottaa kantaa.

Jatkuva integraatio

Jatkuva integraatio toteutui läpi projektin. Käytännössä jokaisen iteraation loppuun-

saattamisen jälkeen sovelluksesta otettiin kopio, jota sitten lähdettiin jatkokehittämään seuraavassa iteraatiossa. Koska sovellus oli toiminnoiltaan kohtalaisen pieni, ei eri osioita kehitetty erikseen, vaan lopputuote oli koko ajan työn alla. Aiemman iteraation kopio oli kuitenkin tallessa siltä varalta, jos jotain peruuttamatonta sattuisi.

Asiakasyhteistyö

Asiakasyhteistyö kehityshankkeen aikana oli pääosin aktiivista. Alussa, ennen kuin XP:n käytäntöjä alettiin ottamaan projektiin mukaan, asiakasyhteistyö oli vähäisempää. XP:n käyttöönoton jälkeen yhteydenpito oli säännöllistä. Toimeksiantajalta saatujen käyttäjätarinoiden perusteella tarkennettiin sovelluksen vaatimuksia ja toimintoja yhdessä. Asiakas oli arkisin tavoitettavissa toimipisteestään, jossa tekijä vieraili tarvittaessa perjantaisten esittelyjen lisäksi. Kasvotusten tapahtuman kommunikaation lisäksi käytettiin sähköpostia. Kommunikointi oli viikoittaista, ei kuitenkaan päivittäistä, sillä tekijä pystyi hyvin rakentamaan sovellusta itsenäisesti vaatimusten pohjalta.

Pariohjelmointi

Koska projektin tekijöitä oli vain yksi, pariohjelmointia ei pystytty toteuttamaan.

Koodin refaktorointi

Koodia pyrittiin refaktoroimaan aina kun mahdollista, mutta viimeistään silloin, kun jokin toiminnallisuus oli toteutettu. Lisäksi koodia refaktoroitiin lisää viimeistelyvaiheessa, iteraatioissa 5-6 (ks. taulukko sivulla 46). Koodia refaktoroitiin esimerkiksi rakentamalla tietokantayhteys toisella tavalla sekä luomalla yleisiä metodeja, jotka olivat käytössä useammassa osassa ohjelmaa. Refaktoroinnin kautta koodia saatiin selkeytettyä ja sen eri osat olivat helpommin erotettavissa.

6 POHDINTA

Ohjelmistoprojektin toteuttaminen ketteriä menetelmiä käyttäen antaa paljon erilaisia mahdollisuuksia niin projektin hallintaan kuin käytäntöihinkin. Perinteinen vesiputousmalli on jäänyt teknisen kehityksen myötä raskaaksi pääasiassa sen huonon reagoimiskyvyn vuoksi. Ketterät menetelmät pystyvät reagoimaan asiakkaan vaatimuksiin ja muihin muutoksiin nopeasti lyhyiden iteraatioiden ja tiiviin asiakasyhteistyön ansiosta. Ketteristä menetelmistä koodaamiseen suunnattu XP soveltuu parhaiten projekteihin, joissa on hyvä tekninen osaaminen ja joissa asiakas on fyysisesti lähellä projektiryhmää ja tarvittaessa käytettävissä. XP:n käytännöt, kuten pariohjelmointi ja testilähtöinen kehitys, vaativat kehittäjiltä hyviä kommunikointitaitoja teknisen osaamisen lisäksi.

6.1 Kokemukset XP-projekteista

Opinnäytetyössä esille tulleissa käytännön ohjelmistoprojekteissa XP:tä on pystytty hyödyntämään hyvin. Kaikkia käytäntöjä ei projekteissa käytetty, mutta molemmissa tapauksissa niistä valikoitiin sellaiset, jotka sopivat juuri kyseiseen projektiin parhaiten. Semper-projektissa projektiryhmä pystyi omaksumaan XP:n käytännöt hyvin, lukuun ottamatta pariohjelmointia ja testilähtöistä kehitystä. Näiden käytäntöjen kokeilemiseen olisi ollut hyvät mahdollisuudet, sillä kehittäjät olivat tietotaidoiltaan yhdenvertaisia. Tässä projektissa projektiryhmä ei ehkä ollut vielä tarpeeksi kypsä soveltamaan näitä käytäntöjä, sillä kaikilla työntekijöillä ei tiettävästi ollut työkokemusta vastaavista projekteista, vaikka heidän taitonsa olivat ajan tasalla.

ThoughWorksin ensimmäinen XP-projekti oli erittäin onnistunut ja etenkin kommunikaatio koko projektiryhmän sisällä parantui huomattavasti. Projektiryhmän roolit oli jaettu hyvin ja rooleja myös vaihdettiin projektin aikana. Myös XP:n käytäntöjä hyödynnettiin ja esimerkiksi pariohjelmointia käytettiin koko projektin ajan, toisin kuin Semper-projektissa. Suuren projektiryhmän koon vuoksi ongelmia ilmeni kuitenkin muun muassa aikataulutuksessa sekä hallinnollisissa asioissa. Ongelmallista oli suunnitella jokaiselle projektiryhmän jäsenelle järkevä aikataulu kuhunkin iteraati-

oon. Projektiryhmän suuren koon ja kehitettävän sovelluksen laajuuden vuoksi projektin aikana oli välillä hankalaa pysyä selvillä siitä, miten projekti tällä hetkellä toimii ja mihin se on menossa.

Yhteenvetona kahdesta esimerkkiprojektista voidaan todeta, että XP:n kurinalainen noudattaminen vaatii tietynlaisen projektiryhmän. Tällaisen projektiryhmän tulisi kooltaan olla pieni tai keskisuuri, sen kehittäjien tulisi olla erittäin osaavia, hyvin kommunikoivia ja halukkaita työskentelemään tiiviissä yhteistyössä asiakkaan kanssa. Myös asiakkaalta vaaditaan tiettyä teknistä osaamista, läsnäoloa ja kokemusta XP-menetelmistä, kuten käyttäjätarinoiden kirjoittamisesta. Käytännössä tällaista projektiryhmää voi olla hankala löytää, joten XP:tä onkin totuttu soveltamaan kunkin projektiryhmän tarpeisiin ja käyttämään sen käytäntöjä esimerkiksi yhdessä jonkin toisen ketterän menetelmän kanssa.

6.2 Pienen ohjelmistoprojektin toteutus

Tekijän toteuttamassa ohjelmistoprojektissa XP-menetelmiä hyödynnettiin niiltä osin, kuin se oli mahdollista. Projektiryhmän pienen koon vuoksi useita XP:n käytäntöjä ei voitu toteuttaa. Ohjelmistokehitysprosessi oli kuitenkin yleisesti ketterä ja lähempänä XP:tä kuin esimerkiksi Scrumia tai Crystal-menetelmiä.

Lindbergin (2003, 42) kokemuksesta iteratiivisissa projekteissa muutosten hallinta on helpompaa ja se on tehokas tapa kohdata puutteellisiin tai muuttuviin määrittelyihin liittyviä ongelmia. Hänen mukaansa asiakkaat ovat jopa itse ehdottaneet iteratiivisten menetelmien käyttöä huomattuaan, etteivät ohjelmiston vaatimukset ole kokonaisuudessaan selkeitä. Tekijän kokemukset opinnäytetyössä toteutetusta ohjelmistoprojektista tukevat Lindbergin näkemystä aiheesta. Tekijän aiemmat kokemukset vesiputousmallin mukaisista projekteista ovat olleet hankalia nimenomaan vähäisen asiakasyhteistyön vuoksi. Iteratiivinen toimintatapa ja läheinen yhteistyö asiakkaan kanssa helpottivat ohjelmistoprojektin läpivientiä etenkin, kun projektin tekijöitä oli vain yksi eikä välitöntä vertaistukea ollut näin saatavilla. Säännölliset, viikon välein pidetyt palaverit, joissa edellisen iteraation tuotos käytiin läpi ja seuraava iteraatio

suunniteltiin, olivat tärkeässä asemassa projektin onnistumisessa. Asiakkaalta saadun palautteen perusteella sovellusta pystyttiin muokkaamaan ja kehittämään oikeaan suuntaan ja näin ollen pystyttiin välttämään turha työ.

XP:n menetelmistä testilähtöistä kehitystä tai pariohjelmointia ei pystytty toteuttamaan. Testilähtöinen kehitys vaatii paljon teknistä osaamista, jotta se voidaan toteuttaa. Ei esimerkiksi riitä, että käytettävä ohjelmointikieli on kohtalaisen hyvin hallussa, vaan sitä pitää osata niin hyvin, että sillä voidaan kirjoittaa testitapauksia. Yhtenä ongelmana testilähtöisessä kehityksessä voi tulla esiin tilanne, jossa etukäteen kirjoitettu testi sisältää virheen. Näin ollen kun toteutettua vaatimusta testataan, se palauttaa aina virheen eikä koodin muokkaaminen tässä tapauksessa auta. Tässä tapauksessa apuun tulee XP:n yksi käytäntö: pariohjelmointi. Toinen parista saattaa huomata virheen testissä, ennen kuin vaatimusta aletaan toteuttaa. Joka tapauksessa molemmilta ohjelmoijilta vaaditaan hyvää ja samantasoista teknistä osaamista. Jotta näitä käytäntöjä olisi pystytty hyödyntämään laajemmin, olisi projektiryhmän pitänyt olla isompi ja sen jäsenten teknisen osaamisen parempaa.

Vaikka opinnäytetyössä ei pystytty käyttämään XP:n kaikkia menetelmiä toisin kuin oli suunniteltu, aiheeseen tutustuminen lisäsi kuitenkin tekijän osaamista tältä osaalueelta. XP:n ideologian mukaisesti sen käytäntöjä voitiin kuitenkin soveltaa projektin tarpeita vastaaviksi, joten kokonaisuudessaan projektia voidaan pitää onnistuneena.

LÄHTEET

Abrahamsson P., Salo, O., Ronkainen, J. & Warsta, J. 2002. Agile software development methods. Reviews and analysis. VTT Publications 478. Espoo: Otamedia.

Aro, J. 2010. Kortteja. Sähköpostiviesti 22.2.2010. Vastaanottaja E. Hänninen. Luettelokortit projektin suunniteluun liittyen.

Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, F., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J. & Thomas, D. 2001. Manifesto for Agile Software Development. Viitattu 10.3.2010. http://www.agilemanifesto.org/.

Beck, K. 2000. Extreme Programming Explained. Embrace change. USA: Addison-Wesley Inc.

Beck, K. & Fowler, M. 2001. Planning Extreme Programming. USA: Addison-Wesley Inc.

Cara, T. & Fowler, M. 2000. An iteration in the life of an XP project. Cutter IT journal. Marraskuu 2000, vol. 13, no. 11. Saatavissa www-muodossa. Viitattu 31.3.2010. http://www.scribd.com/doc/196726/An-iteration-in-the-life-of-XP-project.

Collins, C. & Miller, R. 2001. XP distilled. Viitattu 31.3.2010. http://www.ibm.com/developerworks/java/library/j-xp/.

Haikala, I. & Märijärvi, J. 2004. Ohjelmistotuotanto. Helsinki: Talentum.

Hanselman, S. 2010. Choosing the Right Programming Model. Video. Viitattu 1.4.2010. http://www.asp.net/learn/videos/video-9639.aspx.

Jacko, J. (ed.), 2007. Human-Computer Interaction. 12th International Conference, HCI International. Vol. 1: Interaction Design and Usability. Germany: Springer-Verlag.

Ketterät käytännöt, n.d. Ketterät käytännöt - Menetelmät. Viitattu 30.3.2020. http://www.ketteratkaytannot.fi/fi-FI/Menetelmat/.

Kettunen, J. 2008. Agile-manifesti – Kahdeksan menetelmää sen toteuttamiseen. Tietojenkäsittelytieteen pro gradu -tutkielma. Kuopio: Kuopion yliopisto.

Koskela, L. n.d. Scrum: Ketterien menetelmien markkinajohtaja. Viitattu 30.3.2010. http://ttlry-fi-bin.directo.fi/@Bin/0066563dbeb86251e821ac1d3ab56c50/1269942057/application/pdf/11062393/04_ScrumMarketLeaderOfAgileMethods_handout_LasseKoskela.pdf.

Lindberg, H. 2003. Extreme Programming. Tietojenkäsittelytieteen pro gradu -tutkielma. Tampere: Tampereen yliopisto.

Manninen, L. 2009. Palveluyritykselle soveltuva tuotekehitysmalli. Opinnäytetyö. Lahti: Lahden ammattikorkeakoulu.

Microsoft 2009. .NET Framework Overview. Viitattu 2.4.2010. http://www.microsoft.com/net/overview.aspx.

Microsoft 2010. .NET Framework Conceptual Overview. Viitattu 2.4.2010. http://msdn.microsoft.com/en-us/library/zw4w595w(v=vs.90).aspx.

Peltoniemi, K., 2009. XP vaihtoehtona perinteisille ohjelmistoprosessimalleille. Tietotekniikan pro gradu -tutkielma. Jyväskylä: Jyväskylän yliopisto.

Pohjonen, R. 2002. Tietojärjestelmien kehittäminen Toolkit. Jyväskylä: Docendo Finland.

Sakaryan, G. 2010. What does a ScrumMaster Do? Viitattu 30.3.2010.

http://www.scrumalliance.org/articles/160-what-does-a-scrummaster-do.

Schwaber, K. 2007. What Is Scrum? Viitattu 30.3.2010.

http://www.scrumalliance.org/resources/227.

Schuh, P. 2004. Integrating Agile Development in Real World. Charles River Media Inc.

Serena Software 2007. An introduction to agile software development. Viitattu 10.3.2010. http://www.serena.com/docs/repository/solutions/intro-to-agile-devel.pdf.

Sydänheimo, H-P 2009. Dynaamisen www-sovelluksen toteuttaminen Visual Basic .NET-kielellä. Case: Koekalenteri (Suomen Noutajakoirayhdistys ry). Opinnäytetyö. Tampere: Tampereen ammattikorkeakoulu.

Wells, D. 2009a. Refactor Mercilessly. Viitattu 29.3.2010. http://www.extremeprogramming.org/rules/refactor.html.

Wells, D. 2009b. The Values of Extreme Programming. Viitattu 29.3.2010. http://www.extremeprogramming.org/values.html.

Wells, D. 2009c. Manage by Features. Viitattu 31.3.2010. http://www.agile-process.org/byfeature.html.

LIITTEET

LIITE 1. Tietokonepakettilaskurisovelluksen aloitussivu

LIITE 2. Tietokonepakettilaskurisovelluksen Tilaus-sivu